
S-PM 160400

# Eric

## Python (3 & 2)
## Integrated Development Environment

---

# Technical Report

-<>-

# *Copyright Page*

*S-PM 160400*




*Current edition* PDF file, `Eric_Deliver` site URL: http://www.box.net/shared/k64yenrpey under the "Creative Commons License"

*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

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



– = –

# Foreword

*S-PM 160400*

*Dear Reader,*

*Here is this "*Eric – Technical Report*" to enrich the collection of* Reports *so far produced about this Developing Environment, all under the same basic principles characterizing this whole testing and documentation work [*see on: Eric Web Site*].*
*That is, in summary:*

- *All what you'll read here is the result of a test & documentation project carried on in fair collaboration with the Eric producer but, nevertheless, in total independence from him.*
- *Besides positive technical documentation and instructions, also our possible doubts, perplexities and difficulties have been here fairly reported.*
- *Technical dialogue with you is strongly encouraged, to the obvious benefit of all involved parties. To some extent we'll measure the effectiveness of this* Report *by the amount of feed-back originated, as an evidence of the active role that the user community of such advanced s/w products, as Eric is, should be entitled to play. Active role that we strongly advocate and welcome.*

## *Farewell*

*With this* Report *about last major Eric version No. 6—which blends the formerly distinct ver.s 4 and 5 into this unique one—we take our leave, assuming our test and documentation job as completed. Your feedback welcome. Thank you all.*

*The Author*

– –

## *What & Where in the Internet*

| | | |
|---|---|---|
| *Eric Discussion* | Mailing List: | eric@riverbankcomputing.com |
| *Eric Web Site* | Home: | http://eric-ide.python-projects.org/index.html |
| | Docs: | http://eric-ide.python-projects.org/eric-documentation.html |
| *Eric Reports* | Deliver: | http://www.box.net/shared/k64yenrpey |
| | Specimen ("*AsIs*"): | https://www.box.com/s/52xae8aac52badewvdn0 |
| | Author: | Studio-PM@hotmail.com |

– = –

# {0.Lead} *Essentials*

*S-PM 160400*

Essential reference data, so to precisely identify the product we are talking about, its hosting environment, prerequisites, and execution command.

## *Version Reference*

| *Nr* | *Item* | *Version* | *Note* |
|---|---|---|---|
| 1 | Windows 7<br>Windows Vista Ultimate | `6.1 (7601:SP 1)`<br>`6.0 (6002:SP 2)` | Host Operating System |
| 2 | `\Python34` | `03.04.03` | Interpreter, Host Dir. |
| 3 | Eric | `6.1.4`<br>`(rev. 5830a785550f)` | IDE Application |

***Subject*** "Eric Technical Report", short form: "E-TR"

***Prerequisite*** Python 3 (or/and Python 2), with related PyQt graphic library
[*see:* Prerequisites, *in:* {6.Trail}]

***Execution*** Command: `eric6.bat` (inside: `C:\Python3X\Scripts`)

– –

## *Compatibility with Python ver. 3 or/and 2*

From the current ver. 6, this same Eric IDE is fully Python 3 or/and 2 compatible, both considered as an executing program and as a developing environment. Indeed this same unique Eric IDE can be used with Python ver. 3 only, *OR* Python ver. 2 only, or Python ver. 3 *AND* ver. 2, together. This way offering a unique environment where to attenuate the inconveniences caused by such odd incompatibility between these two consecutive Python versions, and possibly easing the transition between them.

That said, we have here decided to adopt and use Python ver. 3—and the consequently related accessories, such as primarily the related PyQt library—as the only base language for this Report[1], and that for manifest reasons of manageability. Giving thus for granted that a fool-proof compatibility should be experienced in case of adoption of Python ver. 2.

-<>-

1 All users making use of a different choice are kindly invited to offer notice of their experiences.

# Scope of this Report

Declared purpose of this Report is to constitute a comprehensive and reliable guide for the Eric's user community, based upon solid tests and practical experience. And, hopefully, also to represent a base and a stimulus for a positive and fruitful discussion both among users and with the Eric's producer. To the common benefit.

The pursue of such an ambitious purpose doesn't come without the acceptance of some intended limits[2], among which are the exclusion of some Eric subjects as considered off-scope for this Report, and therefore here just overlooked. Namely the Special Features and the Special Details, as hereafter listed [*see*].

– –

## *Special Features*

Some outstanding Eric topics and features will be here only hinted at, as they transcend the scope of this Report, and would possibly require a dedicated paper of their own.

### *Some Little Handy Tools*

*Icon Editor* That is: "a tool to perform icon drawing tasks." [*see command:* `Extras > Builtin Tools > Icon Editor...`];

*Snapshot Tool* A tool to generate screen-shots of the whole screen or of a screen's region with a selectable shape [*see command:* `Extras > Builtin Tools > Snapshot...`];

*SQL Browser* That is: "a tool to examine and execute queries on a database." [*see command:* `Extras > Builtin Tools > SQL Browser...`];

*TR Previewer* A tool to preview translated texts of graphic forms, typically to see how they fit into the hosting space [*see command:* `Extras > Builtin Tools > Translations Previewer...`];

*UI Previewer* A tool to preview graphic forms with selectable styles [*see command:* `Extras > Builtin Tools > UI Previewer...`];

*Code Style Checker*
A tool to check and possibly fix coding style issues; e.g.: naming, documentation and code formatting conventions[3] [*see:* `Project > Check > Code Style...`];

– –

2 Then, of course, there certainly are also some un-intended, unintentional limits. Surely another story...
3 That is, according to the PEP 8 Compliance, Syntax and Tabnanny Checks.

### *Some Other Utilities*

*Version Control System*[4]
: As with menu command `Project > Version Control` [*see*];

*Plugin Add-ons*
: Development techniques, as with menu command `Project > Packagers` [*see*];

*Diagramming*
: As with command `Project > Diagrams` [*see*];

*Qt Forms Designer*
: That is: "a graphical designer for Qt applications." [*see section:* Project-Viewer: Forms*, in:* {4.West}];

*Translations*
: That is: "a tool to load and display Qt User-Interface and translation files for a selected language", with such tools as the Translations Viewer and the Qt Linguist [*see command:* `Extras > Builtin Tools > Translations Previewer...`];

*Web Browser*[5]
: That is: "a combined help and HTML file browser" [*see command:* `Help > Helpviewer...`];

*SSL Certificate Manager*
: A tool aimed at managing the set of Secure Sockets Layer (SSL) Certificates available for Eric and the Eric's web browser when executing secure transactions in HTTPS protocol [*see command:* `Settings > Manage SSL Certificates...`];

*IRC Network Management*
: That is the tool where to manage (add / edit) the set of possibly other IRC Networks here usable, beyond the default one as available to the Eric users community [*see section:* *R-Pane*: IRC*, in:* {5.East}];

*Project Packaging*[6]
: That is: to "Generate a distribution package using cx_Freeze" [*see command:* `Project > Packagers > Use cxfreeze`];

*Eric APIs*
: Eric "Application Programming Interfaces", as with command `Settings > Preferences... - Editor > APIs, Configure API files`, or `Settings > Reload APIs` [*see*];

*Debug Remote, Configurable*

---

4 Actually, about this topic an "Eric 4 – Version Control Technical Report" has been produced, substantially valid for this Eric version too [*see:* Foreword*, What & Where in the Internet*].

5 Actually, about this topic an "Eric 4 – Web Browser Technical Report" has been produced, substantially valid for this Eric version too, just a bit outdated [*see:* Foreword*, What & Where in the Internet*].

6 On this topic a preliminary "Eric (5) Project Packager – Technical Report" has been produced about the limited "`cxfreeze`" technique, as currently used by Eric; now available still *AsIs* [*see:* Foreword*, What & Where in the Internet*]. And then also a "Blueprint" Tech. Note, about the more powerful cx_Freeze "`distutils`" technique, envisioned as a possible enhancement for Eric; available only as private paper, upon request. Anyhow, for what we are concerned, no further work is foreseen for this tool [*see also next:* Project Deployment: a Neglected Feature].

As with commands: `Settings > Preferences... - Debugger` and `Project > Debugger` [*see*];

*Source Text Checks*

Some source code checks on the current Python Project, as with command: `Project > Check` [*see*];

– –

## *Project Deployment: a Neglected Feature*

Looking at such an impressive list of Special Features[7], some of which, as the dedicated *Web Browser*, not so commonly found in other IDEs, here we had been rather puzzled by the practical absence of a feature which, as opposite, is given as fundamental in all other IDEs we know. Indeed, what Borland Delphi calls the "Application Deployment", or Microsoft Visual Studio calls the "Release Build", finds no equivalent here in Eric, but for a [*modest*] front-end to call a [*very modest*] external "`cxfreeze`" script, available as a sub-command of the `Project > Packagers` set [*see, and also the* ***Project Packaging*** *here above, among* Some Other Utilities].

We assume *there is a good reason for that*, a reason that we deem as worth being explicitly recalled.

<!> The point is that *Python is not a compiler, but an interpreter*, and Python programs are assumed to remain at source level, not compiled, nor relocate-loaded. That is, not executed at care of the operating system, but interpreted at care of the very Python interpreter itself, assumed as ubiquitously present in all intended target environments.

Of course we know that this scheme as here depicted is neither complete nor always valid, and that things are not so simple. Indeed we've produced a dedicated Report precisely to examine in some more detail what here has been just hinted at[8].

–<>–

7 So rich and powerful that we're inclined to rate Eric as transcending what is usual intended with the IDE acronym.

8 That is: "Eric (5) Project Packager – Technical Report", now available *AsIs* at the `Eric5-TR Specimen` site [*see:* Foreword, *What & Where in the Internet*].

## *Special Details not Requiring Description*

Lots of Eric minor, or too specialized, or highly intuitive features will not be here minutely described, but left to the user's creative understanding; as it would be highly impractical or simply impossible do to otherwise.

### *Examples:*

Command short-cuts and tool-icons,

such as with the entire Eric Tool Bar, here assumed as not requiring any dedicated detailed description [*see:* Tool Bars*, last section in:* {1.North}].

Too specialized parameters,

such as the esoteric `Outgoing mail server (SMTP)` and `Outgoing mail server port`, as requested for command `Help > Report Bug...`, `Help > Request Feature...`, and configurable on the "`Configure Email`" form of the command `Settings > Preferences... - Email` [*see*].

Similar or identical commands,

such as: `Open...`, `Close`, `Save`; available in different menus and that will be described only once, then simply referred to [*e.g. command:* `File > Open...` *as a reference also for:* `Multiproject > Open...` *and* `Project > Open...`].

-<>-

# Eric's Compatibilities

Here the list and then a detailed description of the rather wide range of Eric IDE compatibilities.

- Platforms: Windows, Mac OS X, Linux / Ubuntu / …;
- Python ver.s: 3, 2;
- Languages
  - Full: Python;
  - Limited: Ruby, then: Perl, Lua, Dmd, /bash, /sh [*see next ref.:* ***Shebang***];
  - Formal: C++, Java, … [*see:* `Text Form (^) Languages`, *in:* {2.Central}];

– –

## True Multi-Platform

Eric is a true multi-platform product, and as ubiquitous as Python. That is to say that, in principle, Eric is equally usable and compatible with all such platforms as: Windows, Mac OS X and Linux; where it operates in the same way, but for some minor and clearly unavoidable differences.

In this Report, however, and for many good reasons, we have chosen to make precise reference only to just one of these platforms: Windows [*see above table:* Version Reference], leaving to each user the freedom of choosing any other compatible environment he wants, along with the consequent burden of taking care of the necessary and, hopefully, easily manageable specific differences.

– –

## Compatibility of Eric ver. 4, 5, 6 with Python ver. 2, 3

Historically the concurrent presence, and widespread use, of the two incompatible ver.s 2 and 3 of the same Python language was the reason of the equally concurrent presence of the two corresponding distinct Eric IDE ver. 4 and 5; which preceded the current ver. 6. And, as a consequence, also the reason why, after the "Eric 4 Report" [*see*], here we are engaged with this other analogous Report, originally centered on Eric 5, now updated to Eric ver. 6. With these specific considerations, though:

- We assume that, with rapidly evolving technologies, it is normal to witness the introduction of new subsequent generations of the same product, as with Python 2 and 3.
- But we also assume as less normal to witness such a long lasting contemporary presence of two successive generations of the same product[9], in a process that looks more like a secession than a transition, as it seems to happen with Python 2 and 3.
- Now with Eric 6—onward—instead of a consequent contemporary presence of two distinct versions for this same IDE, we have got a unified one, fully compatible with both current Python ver.s 3 and 2, considered either separately or together in the same computer system.

9 More than five years, now. A geologic era, with informatics.

### *Remark*

<!> As a consequence we assume that, now on, only the ver. 6 of Eric will be actively updated and developed, as a unique IDE for all current Python flavors. A prospective that here we're glad to adopt when talking of *the* Eric Python IDE, *tout court*; avoiding whenever possible any unnecessary ver. specification.

<.> Note that Eric, besides being multi-platform and multi-version, is also somewhat a multi-language IDE, as it can handle both Python 2, Python 3 and, marginally, also Ruby[10] [*see next* Remark].

– –

## Migration Python 2 → 3

A consequence of the here stated Python 2 & 3 compatibility is that Eric (6) can be used as a "perfect" developing environment where to possibly undertake the burden of migrating a Python 2 source to Python 3 [*ref.: next section*], having both Python ver.s installed on the same computer system. With such a set-up configuration, Eric can offer the choice about which the Python version to be used with any given project, as hereafter described.

### *Remark*

<.> Note that, within this Eric IDE, also Ruby modules[11], with "`*.rb`" file extension, could be interleaved, and also with some debugging feature maintained[12]. Support for other languages, such as: Bash, C++, Java, …, is exclusively formal, being limited to such source text editing aid as the Syntax Highlighting [*see: context command* `Text Form (^) Languages`, *in:* {2.Central}].

– –

### *How to Select the Language Interpreter*

Methods for selecting which the language interpreter for a source text with Eric are hereafter listed, in increasing priority order.

---

10  [*Feature Under Revision*] – Ruby support is comparatively rather limited, in particular for the debugger which was originally developed at its 1.8.*x* version time, then "frozen". Now, with current Ruby ver. 2.1.*x*, it has become well outdated. That's why, after current Eric version 6.1, this feature is meant to be reconsidered; that is:

- either effectively maintained by someone who will take such burden over;
- or discontinued and abandoned with next Eric ver. 6.2.

11 By the way, a suggested Ruby ref. book: "The Ruby Programming Language", by D. Flanagan & Y. Matsumoto, O'Reilly Inc.

12 Though possibly outdated [*see above footnote*].
Anyhow a feature that we're not intentioned here to explore, as assumed off scope for this Report.

- *Property Value* – For an entire Eric Project, by means of the "`Progr. Language`" property value as set at the very project creation or later on, by means of command `Project > Properties...` [*see*]. A property value that will play as a default condition for all the other methods hereafter listed.

– –

- *File Extension* – For each single Python module, by means of the module source file extension: `*.py`, `*.py2` or `*.py3`, and acting on the menu command: `Settings > Preferences... – Python, Configure Python, Source association` [*see*]. This setting would take precedence over the former one.

– –

- *Shebang* – For each single Python module, inserting a Unix-standard "Shebang" encoding-comment as first line, with this syntax, as relevant for Eric too[13]:
  `#! [...] PythonN`
  where `N` = `2` or `3`. In other words, out of a Shebang line, Eric will intercept and read only the language interpreter name. This setting would take precedence over the former two here listed.

## *Remark*

Note that, in itself, the Shebang line:

- has the role of setting the path to the interpreter only for Unix and Unix-like platforms, not for Windows where, under this respect, it is ignored;
- has got also a role with Eric for the selection of the Language Interpreter, as here described; in all platforms, Windows included.

– –

- *Editor flag* – For each single Python module, inserting an Eric specific "`eflag`" (Editor flag)[14] encoding-comment line, positioned *at the end* of any source module files. With this syntax:
  `#eflag: FileType = PythonN`
  where `N` = `2` or `3`. With this setting that takes precedence over all the others here listed.

13 Just for completeness' shake, this the whole list of which other Languages would be here recognized by Eric, besides Python: `Ruby`, `Perl`, `Lua`, `Dmd`, `/bash`, `/sh`. Other not-recognized names will be here simply ignored.

14 Another "*`Keyword = Value`*" pair so far defined for this *Editor flag* mechanism is the "no Quality Assurance":
`#eflag: noqa = M601,M702`
as described at related Code Style Checker command `Project > Check > Code Style...` [*see, and cf. corresponding context:* `Project-Viewer: Sources (^)` *and* `Text Form (^) Check > …`].

Note that, according to such Python 2 – 3 choice, also these other editing aids, as the "Typing Completer" and the "Language Highlighter", will be automatically and consequently selected; as for the setting on: `Settings > Preferences... - Editor > Highlighter > Filetype associations` [*see*][15].

## *Remark*

<!> This way, both Python 2 and Python 3 modules can be freely intermixed within the same Eric Project[16] [*see also:* `Project > Properties... - Mixed programming languages`].

– –

## *How to Select the Language Interpreter on the Interactive Shell*

- *Interactive Shell* – With Eric the available interpreting language can be selected not only within the source text, but also within the Interactive Shell, acting on the related pop-up context menu command: `Shell (^) Start`, on the Bottom Form [*see in:* {3.South}].

– –

## *How to Select the Unicode Encoding*

- *Encoding-comment* – Preference for a Unicode source typing environment, instead of the standard 7-bit ASCII, can be expressed inserting this special Unicode encoding-comment:

  ```
  # -*- coding: UTF-8 -*-
  ```

  as first or second line on a Python module[17]. A standard Python convention, commonly dubbed "magic comment", valid with Eric too [*cf. also:* `Text Form (^) Encodings`, *in:* {2.Central}].

  This way, string literals can be typed as a Unicode text, whereas all other Python reserved words and identifiers must anyhow be restricted to the 7-bit ASCII character set.

– = –

15 With not so relevant consequences though, as Python 2 and 3 source highlighters are actually the same. Here you could assign them just different color code settings.

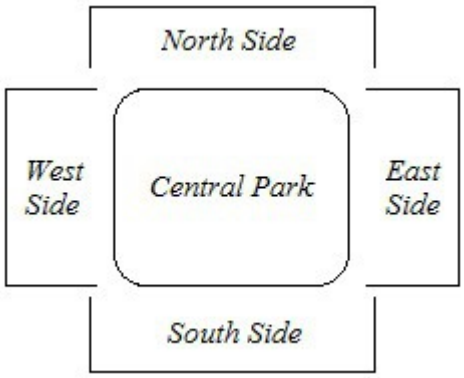
16 And also Ruby modules, with "`*.rb`" file extension. Anyhow a feature that we're not intentioned here to explore, as assumed off scope for this Report.

17 That is, possibly after a *Shebang* line "`#!…`", another conventional encoding-comment well known to all Unix users.

# {1.North} Eric Application Window – *North Side*

*S-PM 160400*

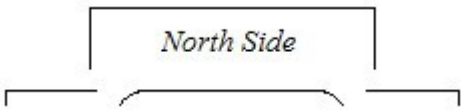
Looking at the Eric Application Window as it were a city-map we can distinguish five main areas:




a *North* and *South Side*, a *Central Park* and an *East* and *West Side* [*cf.:* Eric Window Map*, in:* {Map}*, at the end of this* Report].

– –

This {1.North} section is about the North Side,



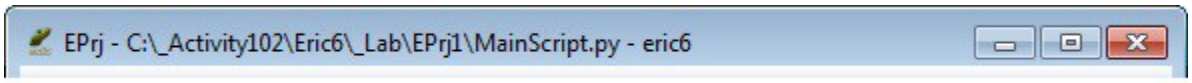

where there are located: *Title Bar*, *Main Menu Bar* with all its commands, and various *Tool Bars* [*see*].

– –

# Title Bar

"`eric6`" Title Bar, positioned, as usual, on top of the application window, with, at left, the standard icon menu and, at right, the Minimize, Maximize and Close buttons.

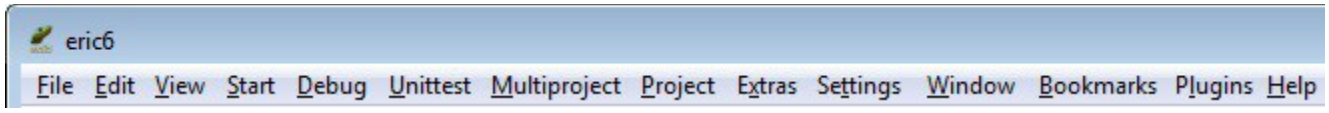

Here shown as in its initial appearance and, hereafter, as with an Eric Project opened, that is with the current module's file path in display.




– –

# Main Menu Bar, and Commands

Here is the Main Menu Bar, in its standard position, just below the Title Bar, and standard shape[18]; where all drop-down Command Menus can be called. It's the main subject of this section.




*Menu Index*

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

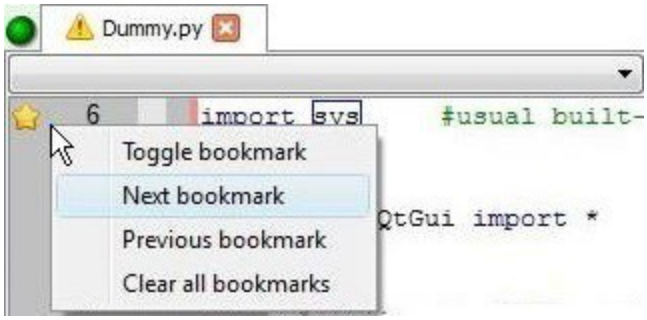
18 <。> That is, exactly as it appears after an original Eric set-up. No custom changes, no options as can be introduced via `Plugins > Install` [*see*] here considered. A usual rule of style, throughout this Report.

Each and all command menus to be thoroughly described hereafter, with the exclusion of such operative details as keyboard shortcuts and Tool Bars, left to the initiative, and preference, of the adventurous user.

– –

## *Remark*

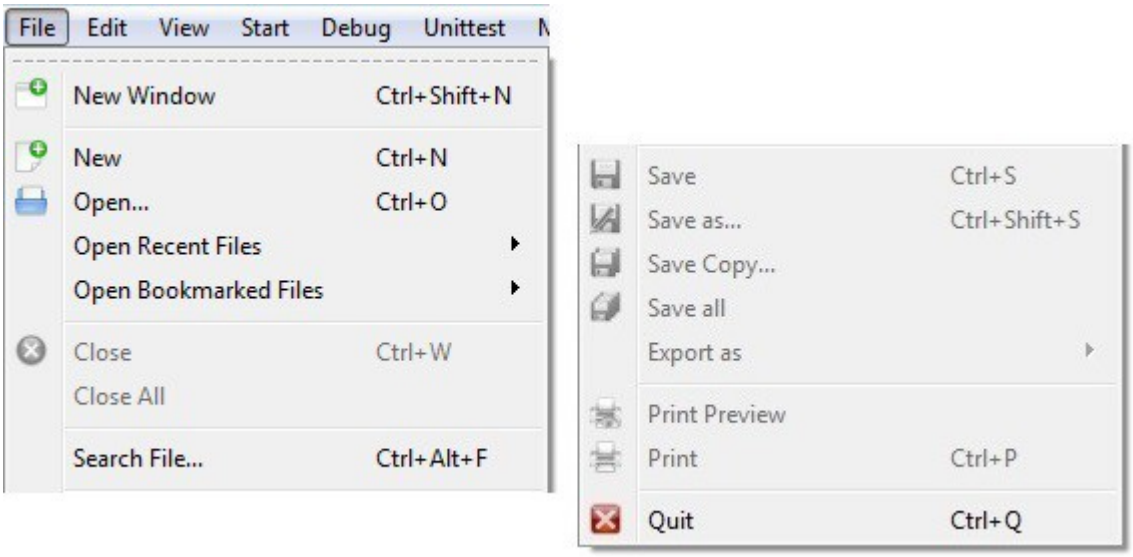
<!> However much important this main menu might be, certainly it doesn't exhaust all the operative power of Eric, which indeed lays in great measure on the many, many pop-up context menus, associated to the various controls of the Eric application window. Pop-up context menus as shown in this example,




conveniently callable with a mouse right-“click”—here symbolically represented with a “(^)”—and as hereafter described in full detail on the subsequent sections of this Report [*see*].

-<>-

# **File** Command Menu




### *Command List*

| | | | |
|---|---|---|---|
| `New Window` | `New` | `Open...` | `Open Recent Files` |
| `Open Bookmarked Files` | `Close` | `Close All` | `Search File...` |
| `Save` | `Save As...` | `Save Copy...` | `Save All` |
| `Export As` | `Print Preview` | `Print` | `Quit` |

– –

## File > **New Window**

Designed to create a new instance of the Eric Application Window. Useful to possibly open a brand new Eric session while keeping the current one still active.

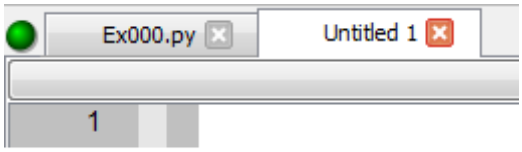
### *Remark*

That is, provided the application is not configured as "`Single Application Mode`" [*see:* `Settings > Preferences... – Application, Configure the application`].

– –

## File > **New**

Designed to create a new *Tagged Text Form* into the Eric *Text* (*Edit*) *Pane*, where to edit source text [*see:* Application Window Map, *in:* {Map}]; with default name: "`Untitled n`".



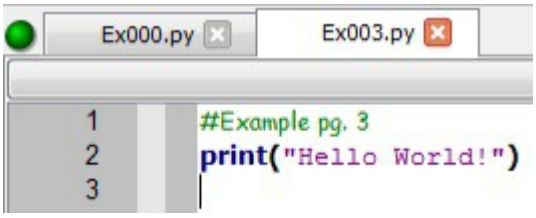

### *Remark*

The contents of such form is meant to be saved as a Python script, that is as a source text file with the typical extension "`*.py`" [*see: command* `File > Save`].

– –

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

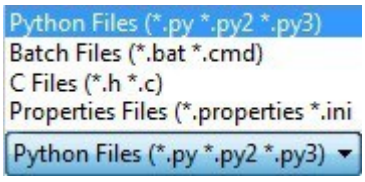
Designed to locate and open typically a "`*.py`" Python source file[19], on a new tagged Text Edit Form, named after the selected file.




This command is exclusively aimed at opening single files, not Projects nor Multiprojects, which have got their own dedicated `Open` commands [*see*].

– –

19 Typically, but not exclusively, as any such text file could be here opened and edited:



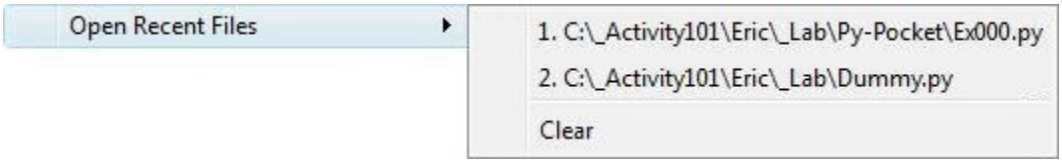
### *Remark*

<!> Both Python ver. 2 or ver. 3 source files (`*.py2` or `*.py3`) can be here opened, edited and properly interpreted, as described at the Eric's Compatibilities section [*see in:* {0.Lead}]. With the initial default file type as set in command `Settings > Preferences... – Editor > Filehandling, Default File Filters, Open Files`[20]. Then with default directory and file extension derived from the tagged Text Edit Form currently active.

– –

## File > **Open Recent Files**

Designed to display the history-list of the last opened files, handy to possibly re-open again one of them [*cf.: command* `File > Open...`].




History-list that here can also be `Clear`-ed.

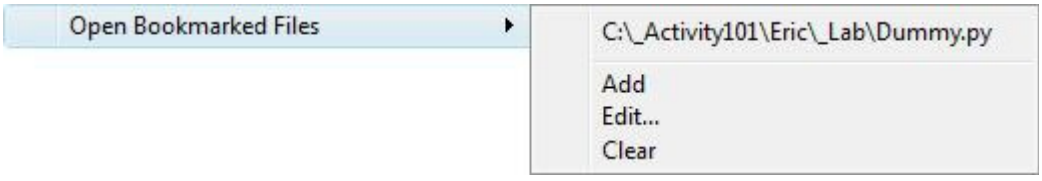
### *Remark*

Maximum length for this history list can be set with command: `Settings > Preferences... – Interface > Viewmanager, Number of recent files` [*see*].

– –

## File > **Open Bookmarked Files**

Designed to show the current list of "Bookmarked Files",




handy to be here opened [*cf.:* `File > Open...`] and managed, with the trailing managing commands:

20 Rather advisable a look there, so to assure your preferred default values [*see also:* Eric Setup Completion, *in:* {6.Trail}].

`Add` To add the currently active file to the Bookmarked Files list;

`Edit...` To show a "`Configure Bookmarked Files Menu`" box, where to edit the list. A box assumed self-explanatory enough not to require any further description[21];

`Clear` To reset the current list.

– –

## File > **Close**

## File > **Close All**

Designed to close the tagged 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.

### *Remark*

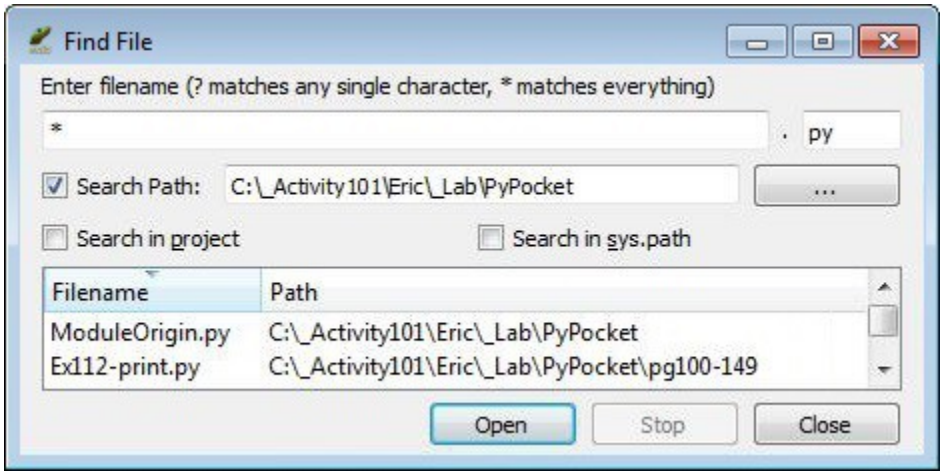
In case of modules belonging to an opened Eric Project [*see: command* `Project > Open`], the project as such remains open.

–<>–

21 But, perhaps, for the `Change` button, aimed at transferring to the selected line item the modifications possibly entered into the "`File:`" field [*see*].

## File > **Search File...**

Designed to show a "`Find File`" dialog box aimed at finding out and then, possibly, at opening files specified by means of usual wildcard characters, and by means of the directory tree branches where to search.




The list of the matching file names is actually displayed only when at least one of the three available search branches — that is:

```
Search Path

Search in project

Search in sys.path
```

has been ticked (i.e: activated), so to tell where to search.

– –

### *Remark*

Be aware that here the "`Search Path`" is taken recursively to all inner sub-directories, as you can see in the above example where the listed files can be found also in sub-directories within the given `Search Path`.

– –

## File > **Save**

Designed to save into the related original file the contents of the currently active tagged Text Form, which then remains open and active [*see: command* `File > Open`]. This command results enabled only after some actual text change.

### *Remark*

<!> Be aware that these `File > Save` commands are intended to be used not only with the very Python modules when opened with the `File > Open` commands, but also when opened as part of an Eric Project, via the related `Project >` or `Multiproject > Open` commands [*see*].

Whereas the corresponding `Project >` or `Multiproject > Save` commands [*see*] have only the much more specialized and limited role of saving strictly the specific data associated with the very collection of Python modules comprising an Eric Multi-Project or Project. That is: *not* the very Python source modules.

– –

### *Viewpoint*

What said in the above Remark implies that Eric makes a distinction between the sheer Eric Project and the set of source modules comprising the same Project[22].

– –

## File > **Save As...**

Similar to the command `File > Save` [*see*], but operating on a new copy of the original source file, which is left unchanged and closed. The new copy becomes the currently open and active one.

This command results active also with unmodified modules, therefore so permitting their duplication [*cf.:* `File > Save Copy...`].

– –

## File > **Save Copy...**

To make a copy of the currently active source module, left anyhow unaltered and open; in that differing from the otherwise similar `File > Save As...` [*cf.*].

– –

22 <~> And if such subtle a distinction comes out a bit baffling to you, be reassured: you are not alone.

## File > Save All

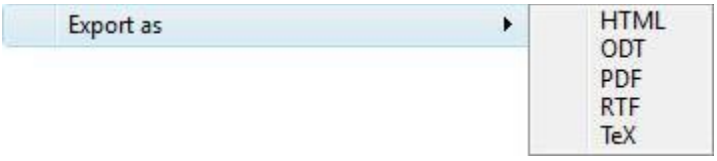
Similar to the command `File > Save` [*see*], but designed to operate onto all tagged Text Forms currently opened.

– –

## File > Export As

Designed to export—that is, to save—the contents of the currently active tagged Text Form into a new file of different format, with the original directory offered as a convenient destination default. The original source remains unaltered.

This the set of available format choices[23]:




| | |
|---|---|
| `HTML` | Hypertext Markup Language, typical of web documents |
| `ODT` | Open Document Text, related to OpenOffice.org |
| `PDF` | Portable Document Format, related to Adobe Acrobat |
| `RTF` | Rich Text Format, related to WordPad |
| `TeX` | Related to LaTeX typesetting system |

– –

## File > Print Preview

## File > Print

Designed to show the Eric specific "`Print Preview`" or "`Print`" dialog box for the edit text file currently active, so to precisely examine how Eric would possibly print it, or to actually print it [*see also:* `Settings > Preferences – Printer, Configure Printer Settings` *and* `Editor, Style, Configure editor styles, Font(s)`].

23 A set of formats with just a "historical" origin.

### *Remark*

Parameter "`Magnification`", in: `Settings > Preferences > Printer – Configure Printer Settings` [*see*], here reveals particularly relevant, as it affects on the actual size and amount of code printed on a page. Most cases a value of `-1` will do, then you may change and tune it to your preference.

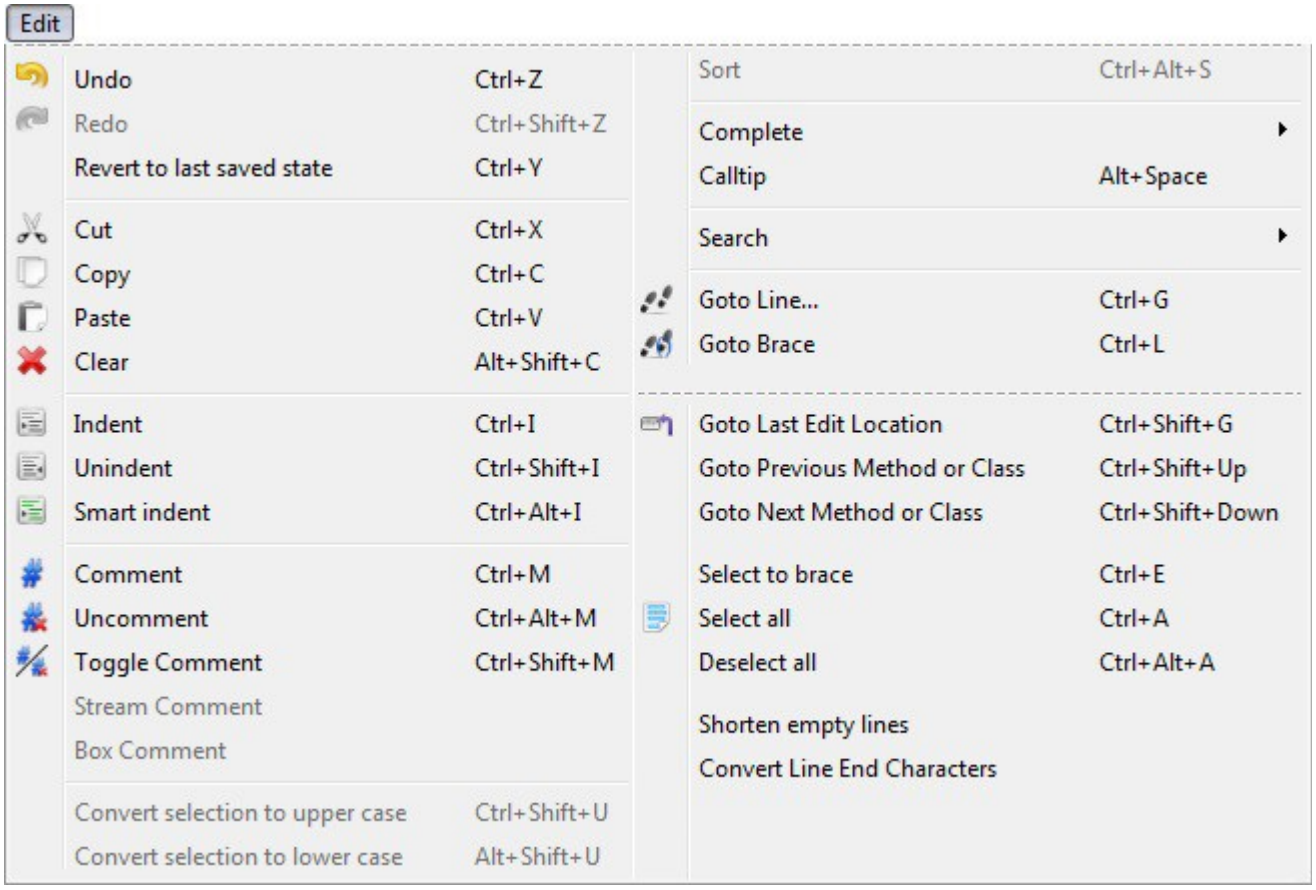
- -

## File > **Quit**

Designed to close the current Eric Window session. In case of unsaved changes, a pop-up "`File Modified`" dialog box will ask the user what do to with them. Other possibly opened Eric Window sessions will remain unaffected [*cf.: command* `File > New Window`].

-<>-

# **Edit** Command Menu




## *Command List*

```
Undo     Redo     Revert to Last Saved State       Cut       Copy     Paste
Clear    Indent   Unindent        Smart Indent     Comment   Uncomment
Toggle Comment    Stream Comment  Box Comment      Convert Selection to Upper Case
Convert Selection to Lower Case   Sort    Complete  Calltip   Search   Goto Line...
Goto Brace        Goto Last Edit Location          Goto Previous Method or Class
Goto Next Method or Class         Select to Brace  Select All
Deselect All      Shorten Empty Lines    Convert Line End Characters
```

– –

## *Remark*

Menu command turned almost completely disabled with focus moved outside the Text Edit Form, in particular even into the "`Find:`" field of the sub-command `Search...` [*see with:* `Edit > Search`].

– –

## 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 currently active 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**

Designed to reset and erase to blank the currently active source Text Form.

– –

## Edit > **Indent**
## Edit > **Unindent**

Designed 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. This same action can be carried on simply making use of the keyboard `Tab` key.

### *Remark*

Default indent value is four blanks, a value changeable via menu command `Settings > Preferences... – Editor > General, Tabs & Indentation` [*see*].

-<>-

## 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.

### *Viewpoint*

<~> We are not sure of having really grasped the logic behind this command, and what to expect from it. Therefore we are not encouraged to using it[24].

– –

## Edit > **Comment**

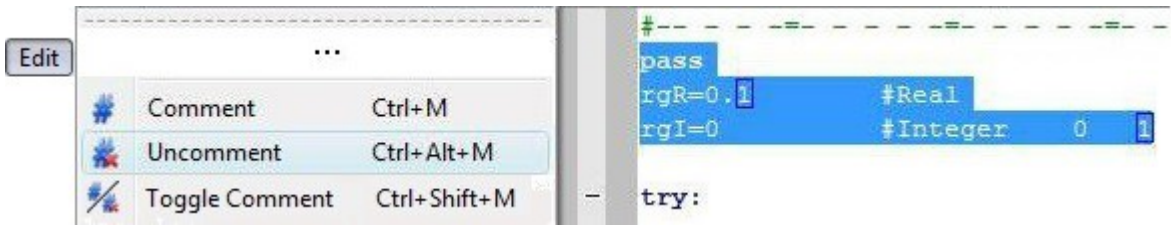
## Edit > **Uncomment**

Designed to add / remove a leading "#"–Python comment character to the pending line, or to a selected block of lines, typically to quickly exclude / include their coding effect at one swoop. The "#" addition will take place also on possibly already "commented" lines; the removal will take effect only at column one, with possibly indented "#"–characters left unaffected.

– –

### *Viewpoint*

<~> As a mildly odd condition you have that selecting such a totally un-commented block of lines, where there is no comment to remove:




the `Edit > Uncomment` command still keeps remaining active; that is apparently usable but, obviously, ineffective, dummy.

– –

24 [*Different opinion and evidence are here welcome*]

## Edit > Toggle Comment

Aimed at conveniently summing-up in one single command both functions of `Edit > Comment` and `Edit > Uncomment` [*see*].

– –

## Edit > Stream Comment

## Edit > Box Comment

<~> A feature related with programming languages other than Python (such as: Ruby or C++), therefore out of the main scope of this Report.

### *Remark*

Eric is potentially a multi-platform, multi-language IDE which, besides Python, could somehow handle other languages such as Ruby [*see:* Eric's Compatibilities, *in:* {0.Lead}].

– –

## Edit > Convert Selection to Upper Case

## Edit > Convert Selection to Lower Case

Designed to convert selected text to upper / lower case. Typically useful to conform global / local variable names to such usual case conventions: `LIKE_THIS` / `like_this`.

– –

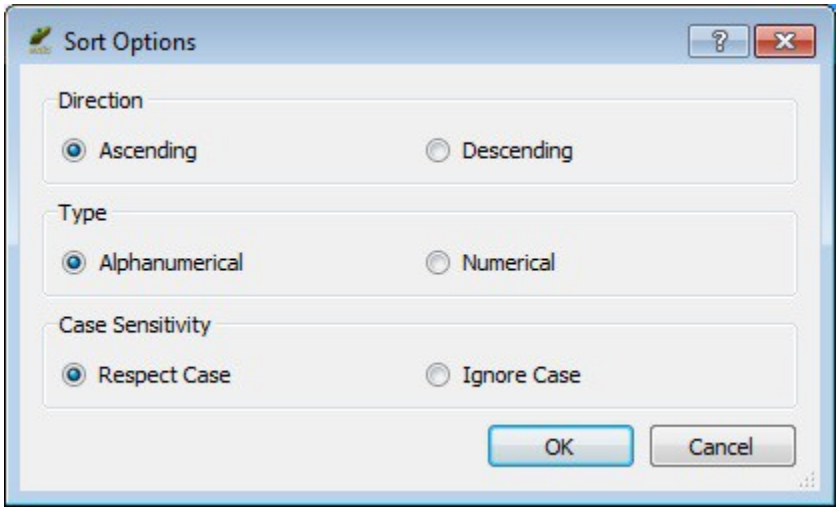
## Edit > Sort

Designed to reorder source text lines according the items located inside a "rectangular" text selection, and according to the control settings on the related "`Sort Options`" form [*see*].

– –

### *Remark*

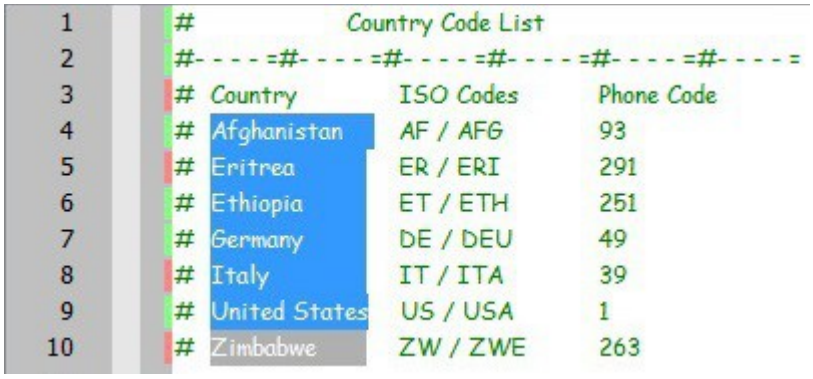
This command results enabled only in presence of a rectangular selection active [*see next:* Rectangular Text Selection *note*].




– –

### *Rectangular Text Selection*

The "rectangular" text selection is a special form of text selection executable with the Eric source editor holding down the "`Alt`" key during an otherwise usual dragging action[25].



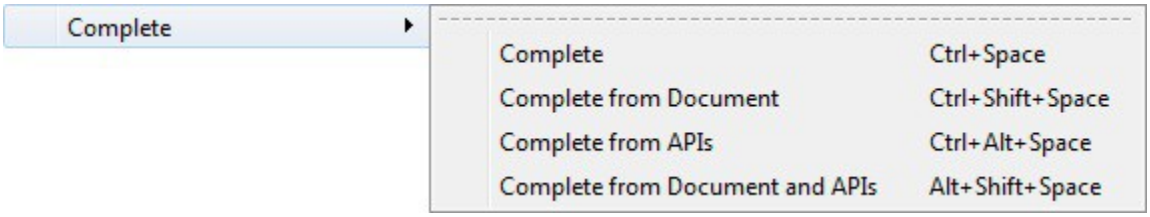

Such feature may come handy with text patterns such as the above "`Country Code List`" example, where an `Edit > Sort` action can be selectively performed on the numeric or alphabetic items, at wish.

-<>-

25 A feature that, where available, may be different, and assume different names; such as: block selection, column selection, non-linear text selection.

## Edit > **Complete**

Designed to show a box of sub-menu commands where to call one of the text *Autocompletion* features available for supporting the source code editing activity [*cf. also: next* `Edit > Calltip` *command*].




– –

### *Preliminary Explanations*

Here some preliminary explanations useful for the comprehension and effective use of this feature, certainly useful, but yet requiring a bit of introduction.

#### *Coding Aids*

As in a word processor you can find such *writing aids* as the spelling and grammar checker for a given human language, so in the Eric IDE you can rely upon such *Coding Aids* as the here available "auto-Completion" and "Call-tip" tools [*see: next* `Edit > Calltip` *command*].

### *Remark*

The `Complete` and `Calltip` features, as here considered, are those specific of QScintilla[26], as available on the Eric's source Text Edit Form. They are available by default after a standard Eric setup, with a behavior that can be then configured and managed via `Settings > Preferences... – Editor > …`; as hinted here and described in detail at the "Eric Setup Completion" section [*see in:* {6.Trail} Setup and General Management].

<。> Other such Coding Aid engines, as the well known Python "Rope"[27], may be added as Eric plug-ins. This a possibility here just hinted at, being assumed as beyond the scope of this Report.

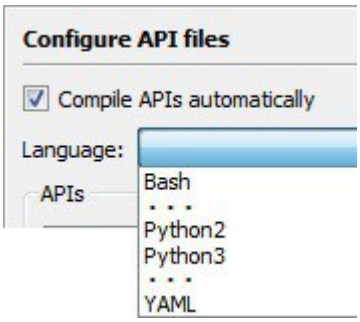
– –

26 QScintilla is a port to Qt of Neil Hodgson's Scintilla C++ editor control, as declared in: http://www.riverbankcomputing.com/software/qscintilla
The original Scintilla is a free library that provides text-editing functions, with an emphasis on advanced features for source code editing, as declared in: http://www.scintilla.org

27 Actually "Rope" defines itself as a Python refactoring library.

### *API (Info) Files*

The Eric Coding Aids, to become fully active, require the preventive selection of which info-resource file will be used to assist your coding activity; that is, the so called API (info) files. This is a preventive selection that has to be made on the `Settings > Preferences... - Editor > APIs, Configure API files` control form, as described at "Eric Setup Completion" section [*see in:* {6.Trail} Setup and General Management].




Here a "`Language:`" choice among a rich list of programming languages for which to activate such code editing aids; from `Bash` to `YAML`, obviously through Python with PyQt[28].

<~> Then you'd better take note of which the language item(s) here chosen as, later on, there is no other way to recall your choice but inspecting on this very form *all* these many items, one-by-one.

– –

### *Viewpoint*

<~> As you see, here in Eric the term "API file", or even simply "API", is often used to signify not precisely a real API s/w package, but all related textual aids instead[29]. A habit that, frankly, we find rather inappropriate and confusing. This is why, for clarity's sake, whenever the case, we'd rather specify "API *info* files" instead of simply saying "API" or "API files".

– –

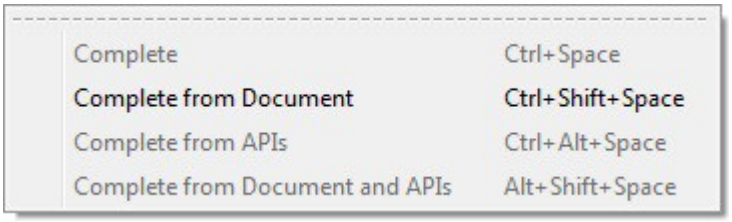
### *Remark*

<!> Note also that at the very first Eric installation, before this configuration, here is how the `Complete` sub-menu will show; that is with most of its commands disabled:

---

28 And also with an `eric6.api` info file, in case you'd like to venture into the development of this very Eric IDE.

29 API (Application Programming Interface) is universally known to signify: "*A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.*" Therefore we feel a bit uncomfortable when, as here, this acronym is used to identify as API something which isn't.
<~> [*if you think that here the "*info file*" specification is useless, as implicit or understood, our opinion and experience is frankly different*].




Then, at all possibly subsequent Eric upgrading set-ups, the formerly enabled configuration will be then kept available[30].

– –

## *Sub-Command List*

`Complete` `Complete from Document`
`Complete from APIs` `Complete from Document and APIs`

– –

`> Complete` Pre-defined auto-completion feature, performing as configured on: `Settings > Preferences... – Editor > Autocompletion > QScintilla` [*see*];

`> Complete from Document`
Text patterns as found on the current source text document;

`> Complete from APIs`
Text patterns as derived from what configured on:
`Settings > Preferences... – Editor > APIs` [*see*];

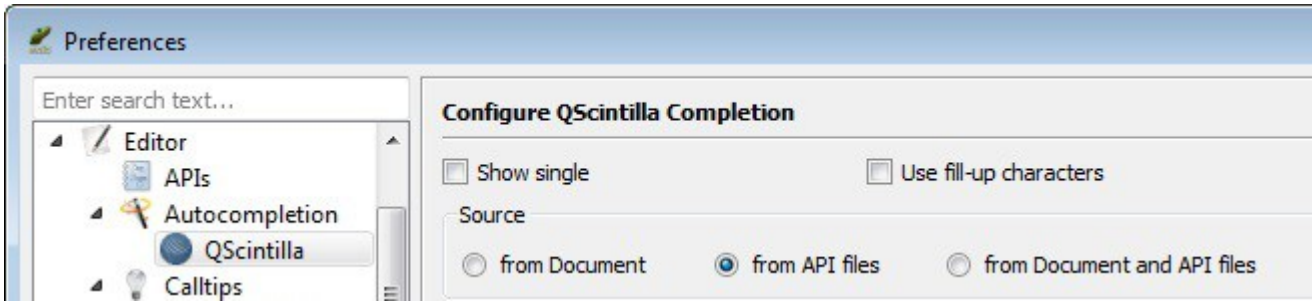
`> Complete from Document and APIs`
A combination of the former two cases.-

– –

### *Keyboard Shortcuts*

Keyboard shortcuts are here rather convenient, particularly the `Ctrl+Space`, for a standard `Complete` action, meant to operate as it has been pre-defined in the `Editor > Autocompletion > QScintilla` preference section, `Configure QScintilla Completion` control form,

30 <。> Unless you'd perform a complete Eric's configuration reset, erasing all "`*.ini`" files recorded into the `<User>\AppData\Roaming\Eric6` directory [*cf.:* Reset User Configuration, *in:* {6.Trail}].

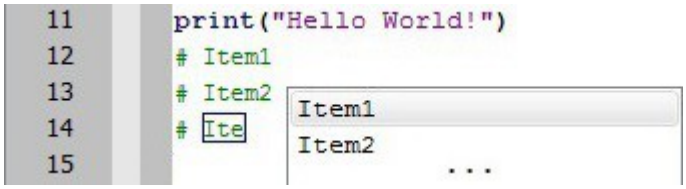



notably in the `Source` area [*via:* `Settings > Preferences...` *command*].

– –

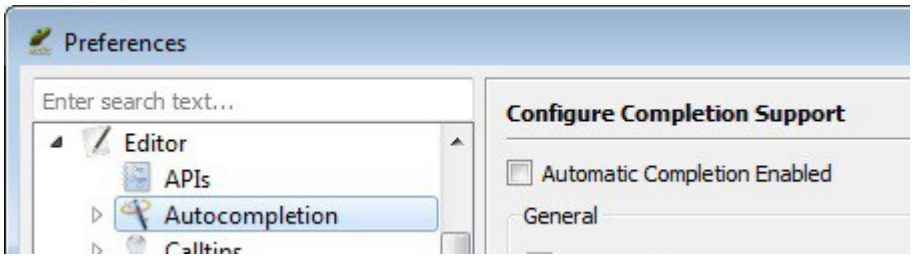
### *Example of Use*

The `Complete` feature is aimed at showing this kind of pop-up choice list:




where a `(^)` "click" (or <`CR`>, or <`Tab`> keystroke) on the selected item can be then entered to possibly confirm a text auto-completion action.

This pop-up list of choices will be shown either upon an explicit manual request, that is entering one of the above `Complete` sub-commands, or automatically, if the "`Automatic Completion Enabled`" check box, on the "`Configure Completion Support`" control form, has been ticked [*see: command* `Settings > Preferences... – Editor > Autocompletion`].




<!> Note that, in case of `Automatic Completion` actually enabled, all `Complete` sub-commands [*see*] can anyhow be called manually too.

-<>-

## Edit > **Calltip**

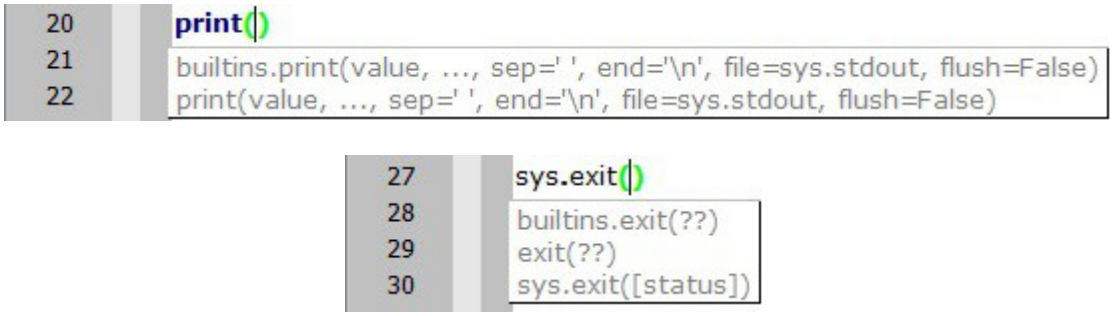
Designed to show the possible signature patterns "`(…)`" of a function, if recognized. A *Coding Aid* command like `Edit > Complete` [*cf.*], also enabled after an APIs configuration as already there described, at the ***API (Info) Files*** section [*see*]; and performing as configured on: `Settings > Preferences... – Editor > Calltips` [*see*].

<。> "`Meta+Alt+Space`" keyboard shortcut is here particularly handy [`Meta` = *Windows logo key* ⊞].

– –

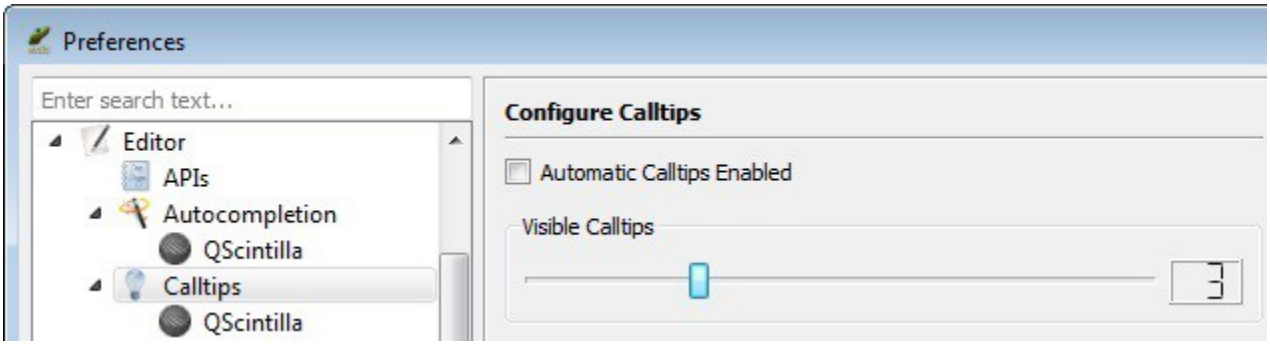
### *Example of Use*

The `Calltip` feature is aimed at showing such function signature's tips[31]:




– –

<。> Tips shown either upon an explicit manual request, entering this very `Edit > Calltip` command, or automatically, if the "`Automatic Calltips`" has been enabled on the "`Configure Calltips`"



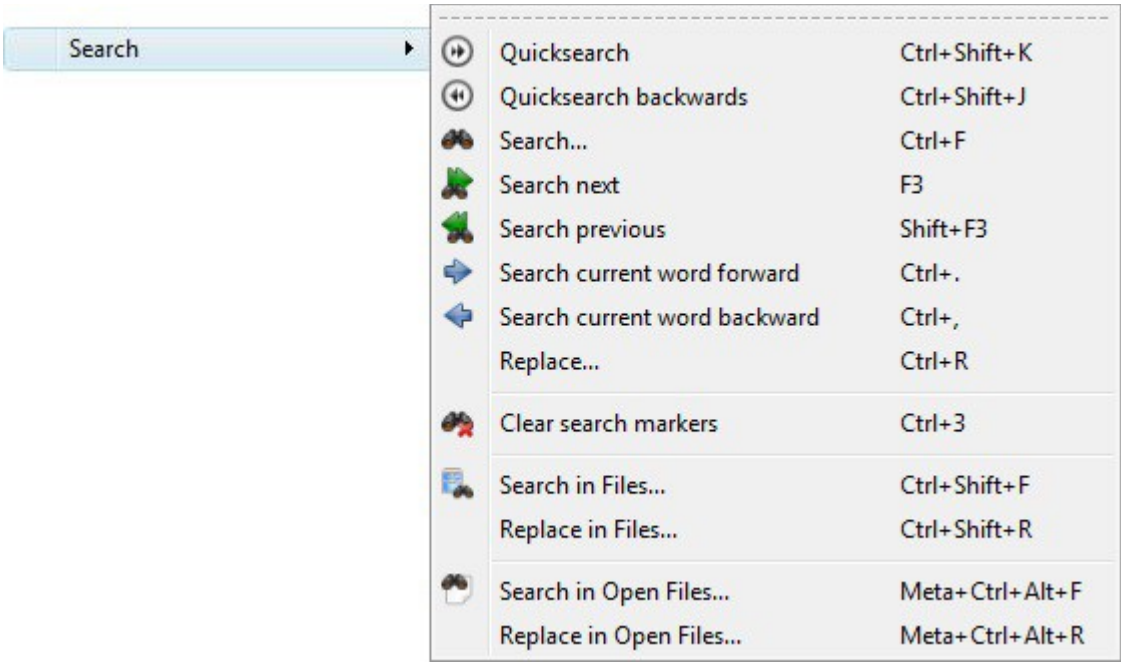

control form [*see: command* `Settings > Preferences... – Editor > Calltips`].

– –

31 <~> Here considered as available after a standard Eric's set-up, no special plug-in nor add-ons; and also without daring a semantic interpretation of the offered tips [*e.g.: no precise idea about the above "??" –Have you any?*].

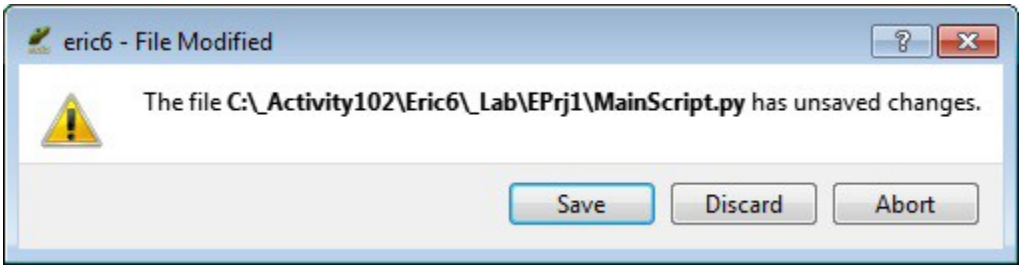
## Edit > **Search**

Designed to show this impressive sub-menu of text search commands.




### *Remark*

Don't be surprised if, activating a `Search`, you are informed that there are "`unsaved changes`".




That's because, as with the `Debug` menu commands [*cf.*], also with these search actions the synchronization with source text must be regained so to have them working properly.

– –

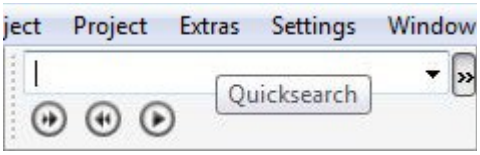
## *Sub-Command List*

`Quicksearch` `Quicksearch Backwards` `Search...` `Search Next` `Search Previous`
`Search Current Word Forward` `Search Current Word Backward`
`Replace...` `Clear Search Markers` `Search in Files...` `Replace in Files...`
`Search in Open Files...` `Replace in Open Files...`

– –

> `Quicksearch` / `Backwards`

To search the currently active Text Edit Form for the first occurrence of a pattern entered into the "`Quicksearch`" field, on the Tool Bar [*see*]; case insensitive, starting from the current text insertion point.




With auxiliary control buttons  : `Quicksearch`, `backwards`, `extend`[32].
Then a `<CR>` on the `Qicksearch` field will bring the insertion point at the end of the string found, selected.

<。> Tool permanently in sight if `Window > Toolbars > Quicksearch` ticked [*see*].

### *Remark*

A pattern not found in the search text is made immediately perceivable, as it cannot be entered in this `Quicksearch` field, with background turned red-colored.

– –

> `Search...` To search the currently active Text Edit Form for the first occurrence of a pattern entered into the "`Find`" field, on the Find Bar located at the bottom of the Text Form.


Available search modes:
`Match case`, `Whole word`, `Regular expression`, `Wrap around`, `Selection only`

– –

32 This option is to extend the quicksearch text and selection to the end of the word currently found, e.g.: QMessageBox

> Search Next / Previous

To repeat the same Search on and on, same search mode [*cf. former point*].
Equivalent to buttons: , and conveniently callable also with the usual F3 / Shift+F3 short-cut.

– –

> Search Current Word Forward / Backward

<。> Super-quick search action, tacking as search-pattern any word currently selected, or even simply pointed at, e.g.: QMessageBox; with short-cut Ctrl+. / Ctrl+, here particularly convenient.

– –

> Replace... To add also a "Replace" text pattern action to the above cited Find field.


With the replace action occurring only after explicit pressing of button: "replace", or "and search for next", or "replace all".

– –

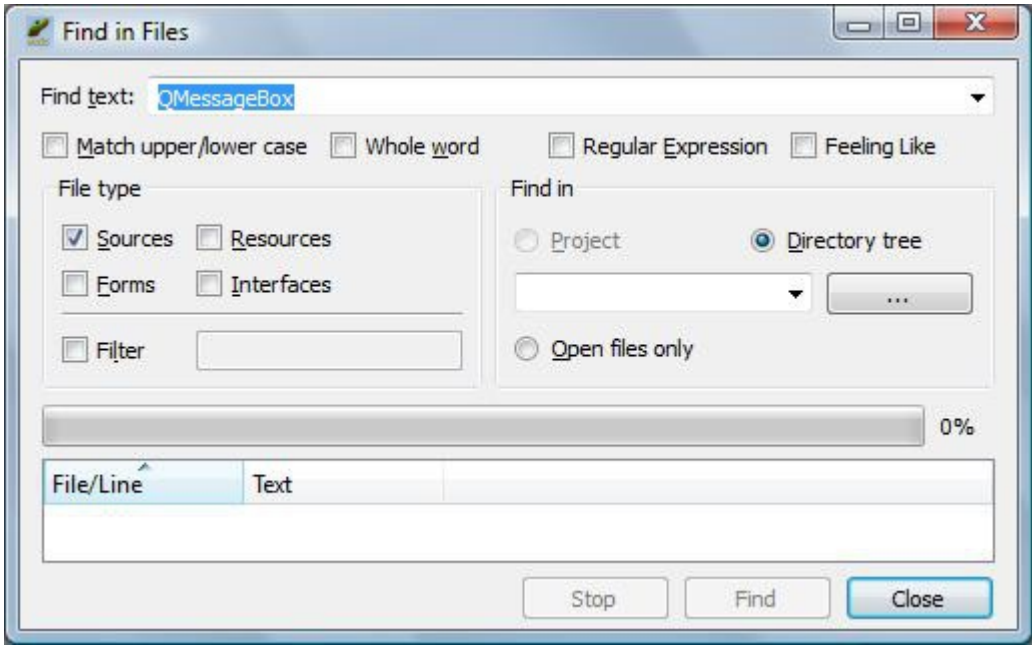
> Clear Search Markers

To clear any such "boxing" graphic markers: marker
possibly left behind by a search action.

– –

> Search in Files...

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




Command particularly useful when working with Projects [*see: menu* `Project`], with this form here assumed clear enough not to require any further explanation.

– –

`> Replace in Files...`

To add a "`Replace text`" field and action to the above "`Find in Files`" dialog box.

– –

## *Remark*

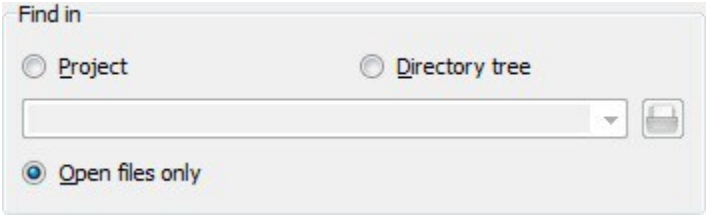
<!> Be well aware that source files possibly used in an Eric Project as Python "`import`-ed" modules may not be considered as part of the Project, in the Eric sense; and, as such, are ignored by the `Search in Files...` command. Whereas they can well be executed and flow-managed with the usual `Debug` commands [*see*], breakpoints included.

So that, to possibly search their text, you must explicitly have them `Open`, and then searched separately from the other Project's files.

– –

`> Search / Replace in Open Files...`

Identical to the above `Search / Replace in Files...` commands [*see*], but for the "`Find in`" option,

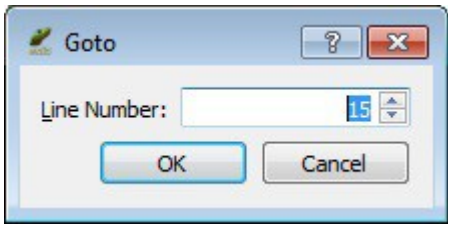



here automatically preset to "`Open files only`".

– –

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

Designed to jump to a given *n*-th line, on the currently active Text Edit Form.




Default "`Line Number`" is that of the currently pending line.

– –

## Edit > **Goto Brace**

Designed to move the text insertion pointer to the next forward, or backward, companion bracket: round `(…)`, squared `[…]`, angular `<…>`, or brace `{…}`; matching that one currently pointed at [*cf. next:* `Select to Brace`]. No action if no bracket is pointed, or within a comment line.

### *Remark*

Some useful hints:

- Matching braces can be highlighted according to command `Settings > Preferences – Editor > Style, Braces` [*see*].
- In the not so infrequent case of such a "`](`" pair of contiguous braces—as in: `_vobjmap[_stype](_ent, _data, _bitpos)`—to reach the matching brace at right of "`](`", you must start from right, instead of from left, as more usual.
- The keyboard shortcut "`Ctrl+L`" in this case is particularly handy[33].

– –

## Edit > **Goto Last Edit Location**

Designed to move the text insertion pointer back to the last edited character, if any. Usage of the keyboard shortcut "`Ctrl+Shift+G`" in this case is particularly handy.

– –

## Edit > **Goto Previous Method or Class**

## Edit > **Goto Next Method or Class**

Designed to move the text insertion pointer to the previous / next Python method or class, if any; otherwise to top / bottom of the document. Usage of keyboard shortcuts "`Ctrl+Shift+↑`" / "`+↓`" in this case is particularly handy.

– –

## Edit > **Select to Brace**

Same as for above command "`Goto Brace`" [*see*], plus the selection of the possibly included text.

– –

33 Just not misinterpret it as a "`Ctrl+Shift+L`", which is to delete lines.

## Edit > **Select All**

## Edit > **Deselect All**

Designed to select / de-select all text on the currently active Text Edit Form.

– –

## Edit > **Shorten Empty Lines**

Designed to clean up all trailing blank characters possibly present on the blank—i.e.: empty—lines of the currently active Text Edit Form.

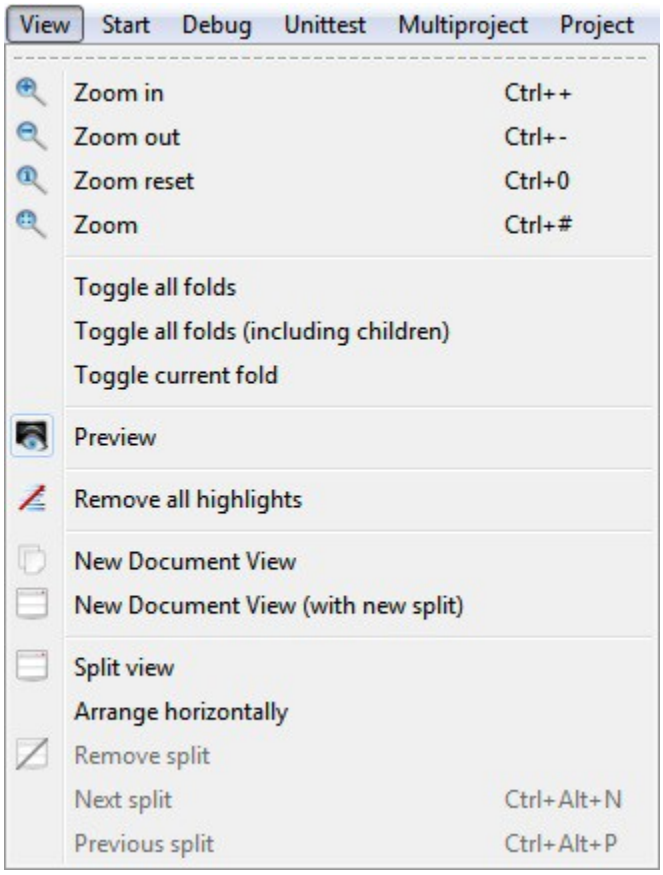
– –

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

Designed to convert all "`End-Of-Line`" characters according to the type currently set with command: `Settings > Preferences... – Editor > Filehandling, End of Line Characters` [*see*].

-<>-

# **View** Command Menu




## *Command List*

```
Zoom In            Zoom Out                Zoom Reset          Zoom
Toggle All Folds   Toggle All Folds (Including Children)       Toggle Current Fold
Preview            Remove All Highlights   New Document View
New Document View (with New Split)         Split View          Arrange Horizontally
Remove Split       Next Split              Previous Split
```

– –

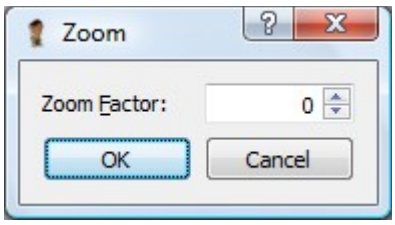
## View > **Zoom In**

## View > **Zoom Out**

## View > **Zoom Reset**

## View > **Zoom**

Commands designed to zoom in, out, reset and parametric (with `Zoom Factor:` 0 = reset) on either the source Text Form or the Python Interactive Shell Form [*see:* Eric Window – *South Side*, *in:* {3.South}]; disabled with focus outside such forms.




Possibly useful to watch a larger amount of long text lines on a crammed display.

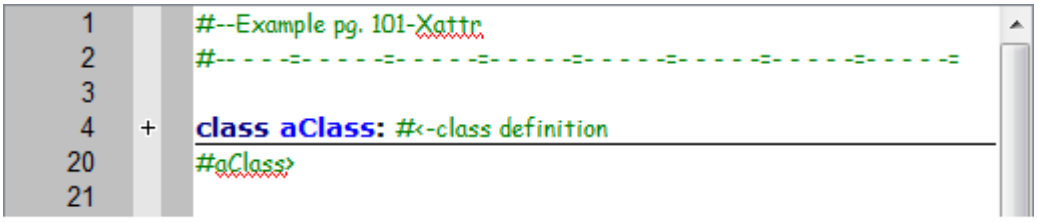
– –

## View > **Toggle All Folds**

## View > **Toggle All Folds (Including Children)**

## View > **Toggle Current Fold**

Defining a "*Fold*" as a source text block positioned at an equally indented level[34], 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.




34 A definition derived from the QScintilla "lexer"—that is, lexical analyzer—as included in PyQt [*cf.:* Prerequisites].

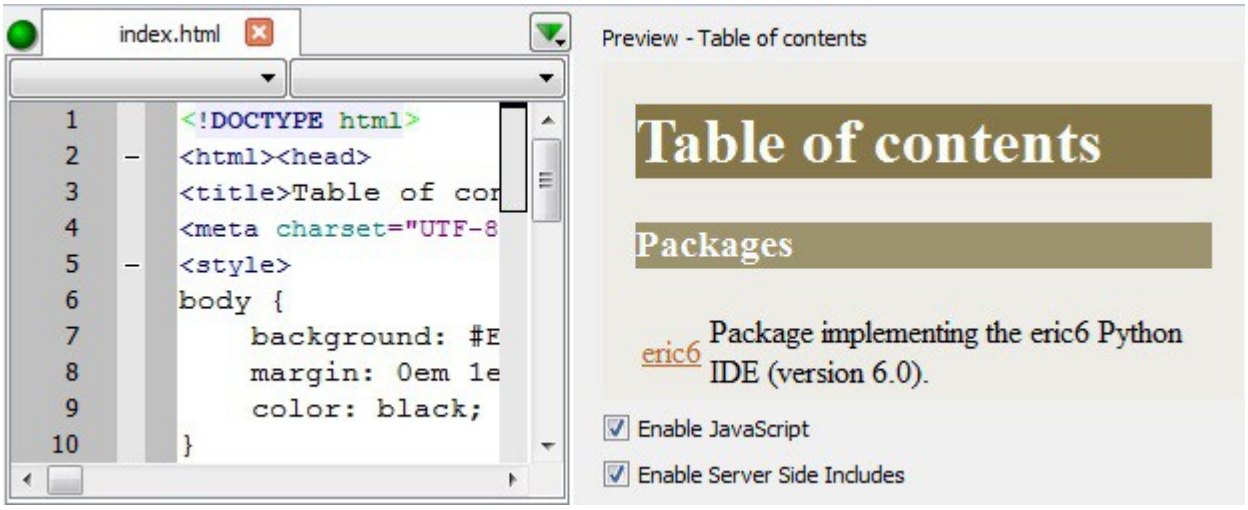
Collapsed folds are indicated by a horizontal line on the source Text Form [*see*], and marked by "+" sign on the vertical Ruler Bar on the left margin; and the expanded folds by a "−" sign[35]. A click on these "+" / "−" symbols has the same toggling effect of these menu commands.

– –

## View > **Preview**

A toggle command to activate / deactivate the internal viewers for these particular text files, when opened in Eric: HTML[36], Markdown, ReST, QSS (Qt Style Sheets);

Display area is a portion of the same Text Edit Form, and the associated file type extensions can be inspected and managed with command `Settings > Preferences... – Editor > Filehandling, File preview` [*see*].




In the above HTML example you have, as associated viewer, the very same Eric web browser [*cf.:* `Help > Helpviewer...`].

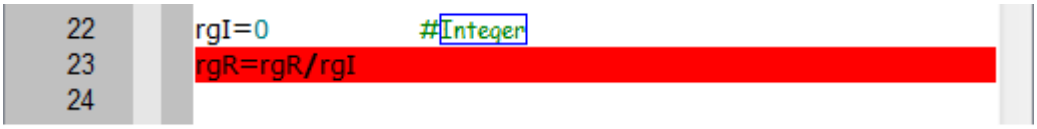
– –

35 Actually these "+" / "−" symbols are configuration dependent, as "`Folding style`" on the "`Margins`" area of the "`Configure editor styles`" [*see:* `Settings > Preferences... – Editor > Style`]. What's here shown is the default "`Plain`" style [*default, indeed: as usual in this* Report].

36 Typically handy to inspect an "EricDoc" documentation file [*cf.:* `Project > Source Documentation > Generate Documentation`].

## View > **Remove All Highlights**

Designed to eliminate all special font-effects on the source Text Form, such as this red-highlight for the exception rising lines.

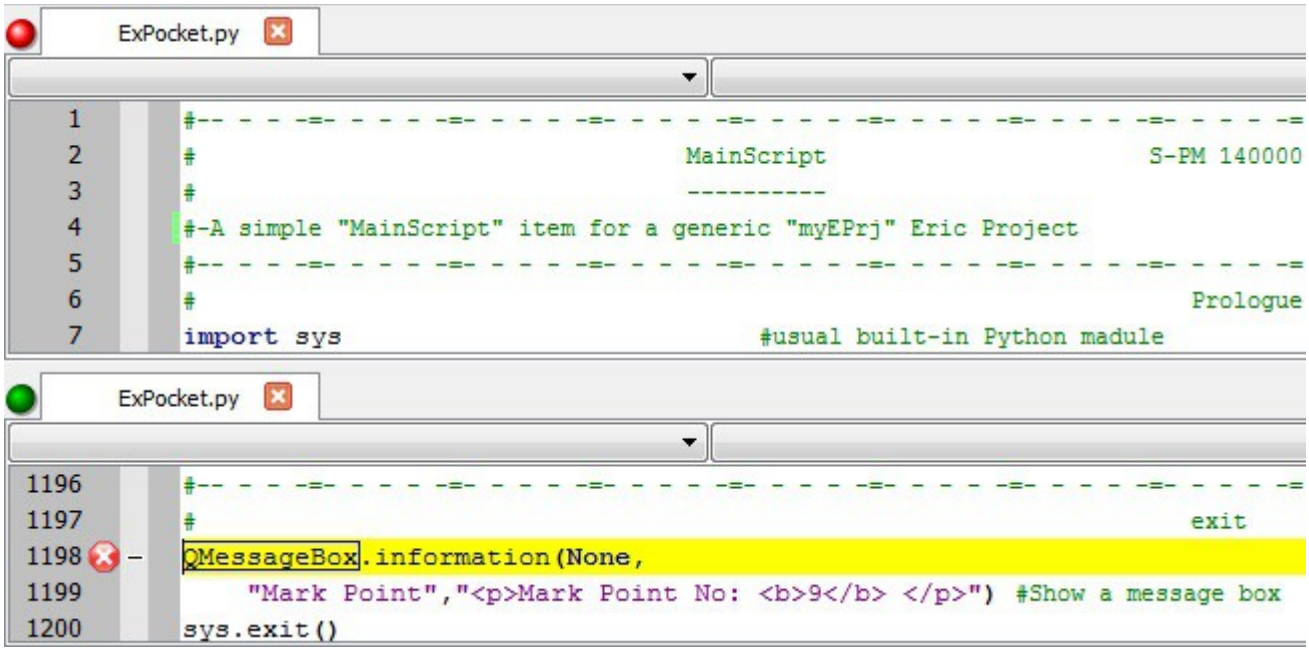



– –

## View > **New Document View**

## View > **New Document View (with New Split)**

Designed to create a new tabbed or split view [*cf.: next* `View > Split View`] of the same source document currently active [*cf. also:* `View > Arrange Horizontally` *hereafter*].

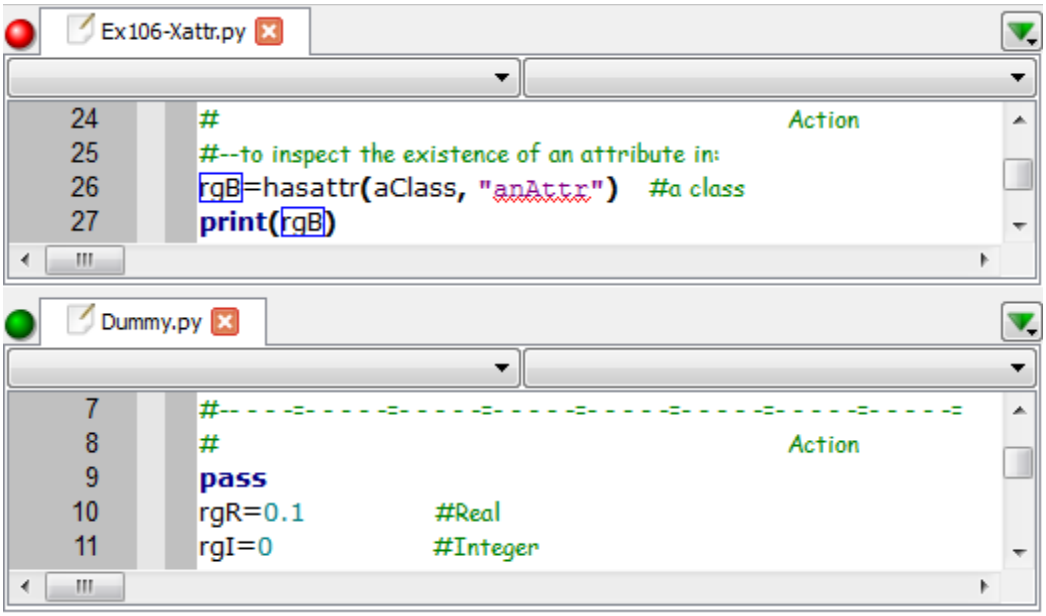



Typically useful for watching or editing "distant" sections of the same long module, such as in the example here shown.

– –

## View > **Split View**

Designed to create a new Text Form split into the same source Text Pane, aimed at opening [*see: command* `File > Open...`] and displaying together distinct sets of source modules into distinct sets of overlapping tagged Text Forms [*see also:* `Remove Split`].

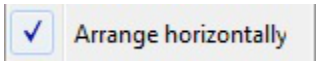



Split views can be arranged horizontally or vertically with following `View > Arrange Horizontally` toggle command [*see*]. The currently active Split Form is marked with a green LED head-pin, all the others with a red one.

– –

## View > **Arrange Horizontally**

Designed to toggle the split view mode [*see: command* `Split View`] from vertical to horizontal, and vice-versa.




A tick-mark appears on the menu at the left of this command, when set.

– –

## View > **Remove Split**

Designed to act on the current Split Form possibly created [*see:* `View > Split View` *command*], first closing all of its Text Forms as with command `File > Close All` [*see*], and then removing the so emptied Split Form from the Central Pane.

– –

## View > **Next Split**

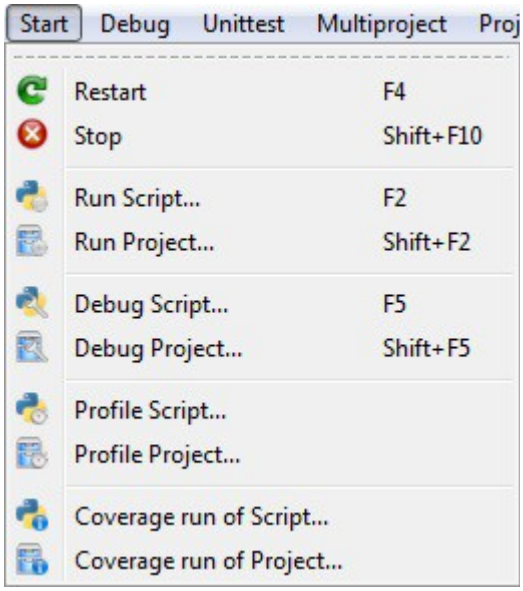
## View > **Previous Split**

Designed to navigate amongst the different Split Text Forms possibly present on the Text Pane [*see:* `View > Split View`]. Same action can be obtained simply clicking on the desired Split Form.

Currently active Split Form is marked with a green LED head-pin  , all the others with a red one  .

-<>-

# **Start** Command Menu




### *Command List*

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

– –

## Start > **Restart**

## Start > **Stop**

Designed to re-start (i.e.: re-initialize and run) / stop the same current execution, whatever it was, either of a script or of a Project[37], and in Run or Debug execution mode [*see:* `Start > Run`].

### *Remark*

When debugging, it is rather normal to modify the executing source code, and then `Restart` it over. Of course, to have the last edited source code executing, you need first to save it, with such command as: `File > Save` [*see*].

37 By the way it is here worth recalling that whereas *script* is a standard Python term, *Project* is a specific Eric term, with no precise correspondence in the Python lexicon.

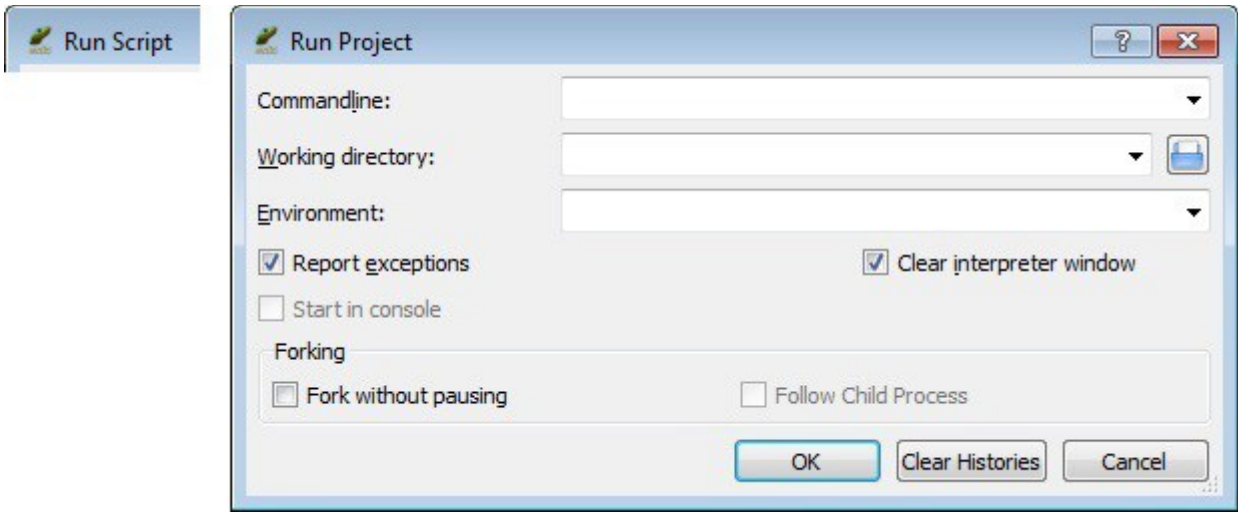
It's here worth knowing that this is an action that can be appointed automatically, with command: `Settings > Preferences... - Debugger > General, Configure general debugger settings, Start Debugging`, check-box: "`Autosave changed scripts`" [*see*].

– –

## Start > **Run Script...**

## Start > **Run Project...**

Designed to start execution of the currently active Python script [*see: command* `File > Open...`], possibly belonging to a Project, or of the current Project [*see: command* `Project > Open...`], with initial conditions that can be set on the specific "`Run Script` / `Run Project`" dialog box.




### *Remark*

It may be useful here to recall that a module 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__')`

– –

### *Specifications:*

`Commandline:` List of arguments to be possibly entered into the run command-line, and retrievable via the standard Python "`sys.argv`" call. To the first argument of this list is automatically assigned the executing *Main Script* file path[38] [*see also: next param.* `Working directory`].

`Working directory:`
To set the initial working directory, as for standard Python "`os.path.abspath (os.curdir)`". Default value being the directory of the executing Main Script file [*cf.: preceding param.* `Commandline`].

`Environment:` To set the value of possibly new environment variables, on the standard Python dictionary "`os.environ`" (e.g.: `USERNAME=`*`myName`*`, NEWVAR=`*`aValue`*).

`Report exceptions`
Check box here ignored, in the sense that in this "`Run`" case, differently from the "`Start > Debug`" case [*see*], the handled exceptions are never signaled; whereas unhandled exceptions are always signaled anyway.

`Clear interpreter window`
To reset what here elsewhere is usually called the Interactive Shell Form [*see:* Eric Window – *South Side*, *in:* {3.South}].

`Start in console`
Enabled only with "`Console Debugger`" enabled, in the "`Configure general debugger settings`" control form [*see: command* `Settings > Preferences... – Debugger > General`]. Otherwise disabled.

`Fork without pausing`
In case of a process fork [*ref.: Python process management,* `os.fork()` *function*], the user is not asked which path of execution to follow [*see: next* `Debug Child Process`].

`Follow Child Process`
Upon a process fork, when ticked, the execution will proceed into the child process, instead of remaining in the parent [*cf.: above* `Fork without pausing`].
Same behavior with `Start > Debug` command too [*cf.*].

`Clear Histories`
A button to clear all history values saved in this form, but for the last value entered on each field, conveniently kept for the next execution.

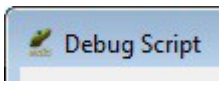
– –

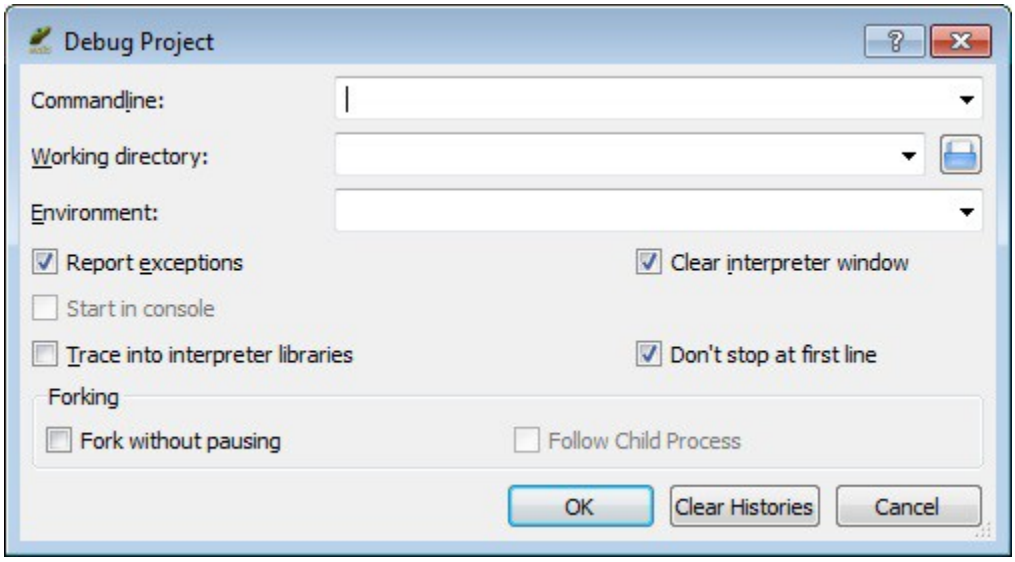
38 About the *Main Script* concept and role, see next section: Project Command Menu

## Start > **Debug Script...**

## Start > **Debug Project...**

Designed to start execution of the currently active script, possibly belonging to a Project, or of the current Project, as with `Start > Run` commands [*see*], but in "debug" mode. That is, activating all tools as with menu `Debug` [*see*], and with initial conditions that can be set on this specific `Debug Script` / `Debug Project` dialog box.




*Specifications:*

[. . .] Besides what already seen with command `Start > Run` [*see*].

`Report exceptions`
Checked to get an Eric dialog box [*cf.:* `Debug > Exceptions`] at both un-handled or handled exceptions — i.e: exceptions occurring within a "`try: …`" block. When unchecked the exceptions are still effective, but no "`Report`" box will be shown.

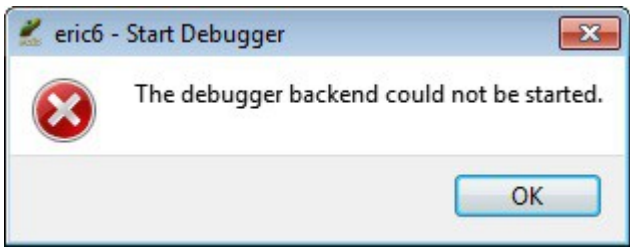
`Trace into interpreter libraries`
Checked to activate tracing debug execution also within standard library modules; otherwise stepped over, as usual.

`Don't stop at first line`
Unchecked to begin execution with a stop at the first executable statement[39], waiting for user's intervention.

39 Note that here a possibly initial "`pass`" statement is ignored by Python, and not taken as an executable instruction. <~> We do not agree.

### *Remark*

In case of such an Error Box:




just a look at the "`Python[N] Interpreter for Debug Client`" area, via command `Settings > Preferences... – Debugger > Python[N], Configure Python[N] Debugger`, may be the right thing to do [*see also:* Eric Setup Completion, *in:* {6.Trail}].

– –

## Start > **Profile Script...**

## Start > **Profile Project...**

Designed to start the execution of a script, or a Project, exactly as for `Start > Run` commands [*see*], but also with the activation of the *Profiling* back-end[40]; so to collect and record the execution profiling data on a disc file[41], examinable later on using one of the following ways:

`Project > Show > Profile Data...`
: Menu command [*see*], for a whole project;

`Text Form (^) Show > Profile Data...`
: Right click context menu, for a particular module on a Project-Viewer form [*cf. in:* {4.West}];

`(^) Show > Profile Data...`
: Right click context menu on the source Text Form, for a single script [*see*].

-<>-

40 Here available as a handy interface to the standard Python `profile` / `cProfile` modules.
41 A "`*.profile`" file, on the same Project's directory.

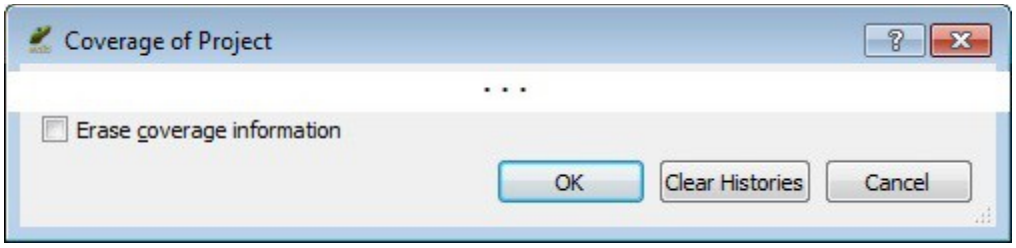
## Start > **Coverage Run of Script...**

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

Designed to start the execution of a script or a Project exactly as for `Start > Run` commands [*see*], but also with the activation of the *Coverage analyzer* back-end[42], so to collect and record the code "coverage" data on a disc file[43]. That is, the data telling which part of the source code has been actually executed and, therefore, also tested during a run session. Comments and empty lines are ignored[44].

– –

All fields on this `Coverage` control box are as explained with the `Start > Run` control box [*see*], but for:




`Erase coverage information`

Check-box, where you have the choice whether to keep or erase the coverage data possibly already collected with a preceding run. Consequences:

- To keep this information means to "OR" the new coverage status condition to all source statements. This implies that the so far un-covered (i.e.: not-executed) statements can possibly switch their status to covered.
- To erase this information means to get a coverage status exclusively as the result of the last run. An advisable choice in case of source changes[45].

– –

42 <。> `Coverage.py`, by Ned Batchelder. A popular tool automatically embedded into Eric, and defined by its author as: "*a tool for measuring code coverage of Python programs. It monitors your program, noting which parts of the code have been executed, then analyzes the source to identify code that could have been executed but was not. – Coverage measurement is typically used to gauge the effectiveness of tests. It can show which parts of your code are being exercised by tests, and which are not.*" [*see URL:* http://nedbatchelder.com/code/coverage/]

43 A "`*.coverage`" file, on the same Project's directory.

44 <~> Whereas, as already spotted [*cf.: similar notes at commands* `Start > Profile` *and* `Debug > Toggle Breakpoint`], here too the treatment of "`pass`" statements is rather ambiguous. Anyhow that's a Python matter, not Eric's.

45 But deliberately here not defaulted this way, though, as with coverage it is more frequent the case of stable source code and variable test data, than the other way round.

Coverage data can then be then inspected by means of:

`Project > Show > Code Coverage...`
: Menu command [*see*], for a whole Project;

`(^) Show > Code Coverage...`
: Right-click context menu command, for a module on the Project-Viewer form [*cf. in:* {4.West}];

`(^) Show > Code Coverage...`
`(^) Show > Show / Hide Code Coverage Annotations`
: Right-click context menu commands on the source Text Edit Form, for a single script [*see*].

`Bookmarks > Next / Previous Uncovered Line`
: Source text edit menu commands possibly activated by a `(^) Show > Show Code Coverage Annotations` [*see*].

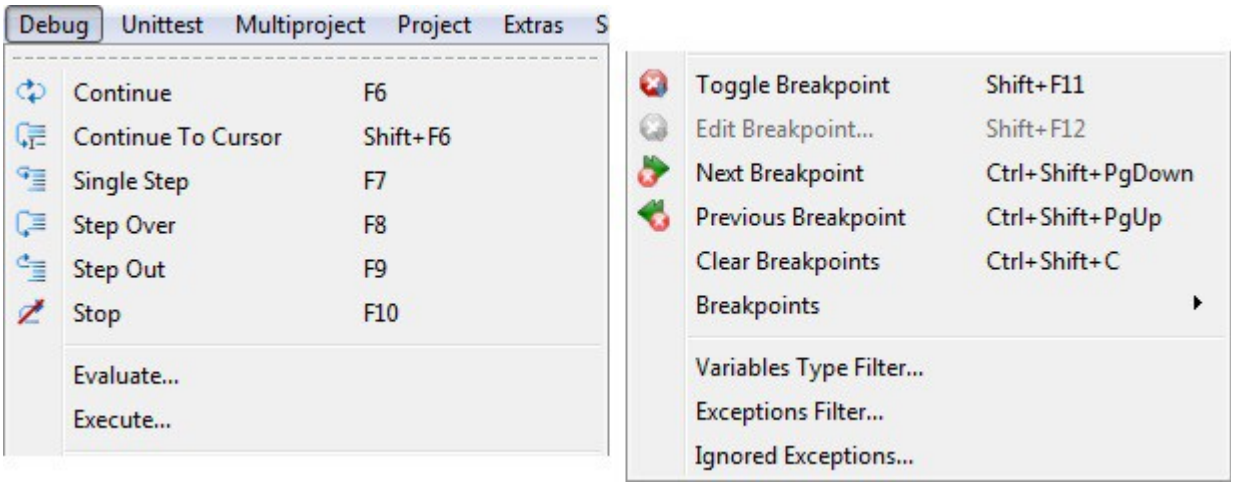
– –

## *Remark*

As for command menu `Unittest` [*see*], this is a tool for the software quality assurance.

-<>-

# **Debug** Command Menu




*Command List*

```
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...
```

– –

## Debug > **Continue**

## Debug > **Continue to Cursor**

Designed to resume execution after a breakpoint up to the possibly next one, if any, or up to the currently pointed statement, excluded.

– –

## Debug > **Single Step**

## Debug > **Step Over**

## Debug > **Step Out**

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

– –

## Debug > **Stop**

Designed to terminate current execution.

– –

## Debug > **Evaluate...**

## Debug > **Execute...**

Designed 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 [*cf.:* `Window > Bottom Sidebar`]. Same action can be carried on operating directly into the very Interactive Shell Form[46].

Note that if your purpose is simply to monitor the value of a variable, you don't even need to use these commands, as you have it shown on the `Global` or `Local Variables` forms of the `Debug-Viewer` [*see:* `Window > Debug-Viewer`].

### *Remark*

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>`"[47].

Indeed, such an identifier, when used outside its class and outside its Eric Text Form, should be written as: "`_<className>__<privateMember>`", otherwise will result unknown. The same happens within the "`Debug-Viewer`" form [*see*].

– –

46 Where no operative difference have we found, but the fact that working on the Interactive Form is easier.
47 Note the "__" prefix convention, characterizing the Python "*private*" items.

## Debug > **Toggle Breakpoint**

Designed to insert / cancel an execution breakpoint on the currently pointed source statement. Same action can be carried on just clicking 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 [*cf.:* `Edit Breakpoint...`].


### *Remark*

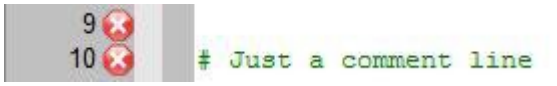
<!> It's worth knowing that the set of breakpoints possibly defined when working on a Project can be saved and then restored making use of the `Project > Session` commands [*see*].

– –

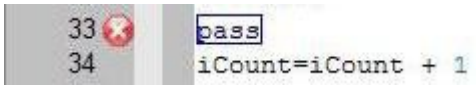
### *Viewpoint*

It is advisable not to rely upon "ghost" breakpoints, that is breakpoints defined onto source text lines that, for some reason, Python considers as not-executable. Such as:

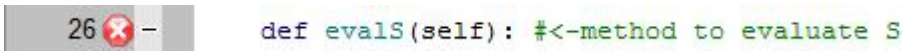
- Empty, or comment lines.




- "`pass`" statements, as they are execution-ignored by Python[48].




- "`def Function(arg)`" statement which, after the initial opening of the source text, will be then ignored by Python when actually calling that function.

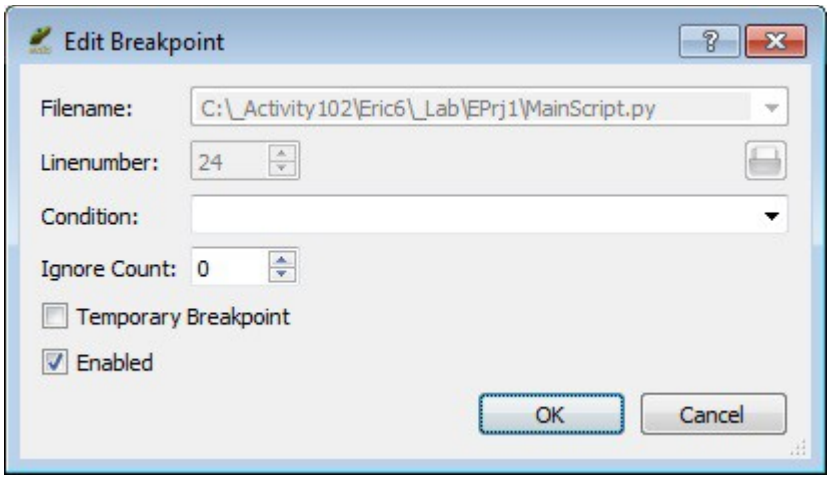



– –

48 <~> We do not agree, as we assume "`pass`" is a statement as any other, precisely as zero is a number as any other.

## Debug > **Edit Breakpoint...**

Designed to show an "`Edit Breakpoint`" dialog box about the the currently pointed breakpoint-line. Command disabled with no such line pointed or selected.




*Specifications:*

`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**

Designed to move cursor to the next / previous breakpoint on the current source Text Edit Form.

– –

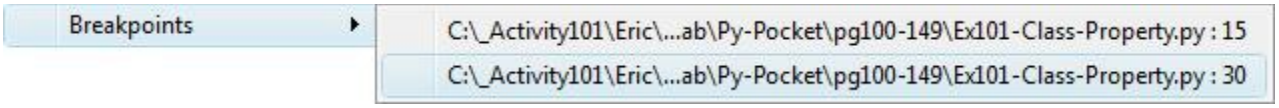
## Debug > **Clear Breakpoints**

Designed to clear all currently defined breakpoints.

– –

## Debug > **Breakpoints**

Designed to display a location list—that is: Python module file path and line number—of all currently defined breakpoints,

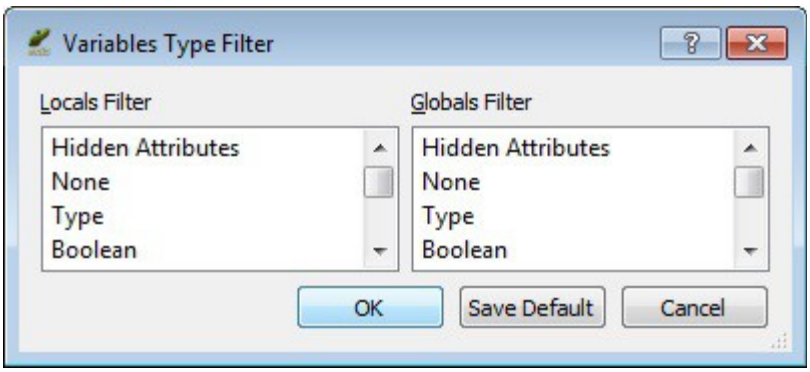



so also to possibly move the cursor onto one of them.

– –

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

Designed to show a "`Variables Type Filter`" box where to select which type of variables, either Local or Global, should *not* be displayed on the `Debug-Viewer` Auxiliary Form; tabs: `Global / Local Variables` [*see:* Application Window Map, *in:* {Map}].




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

– –

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

Designed to show a dialog box where to enter the code of which one of the Python handled exceptions[49] (such as: `TypeError`, `ZeroDivisionError`, …) are 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 Python un-handled exceptions remain unaffected.

Command aimed at focusing debug on chosen exceptions only [*see also:* `Ignored Exceptions...`].

– –

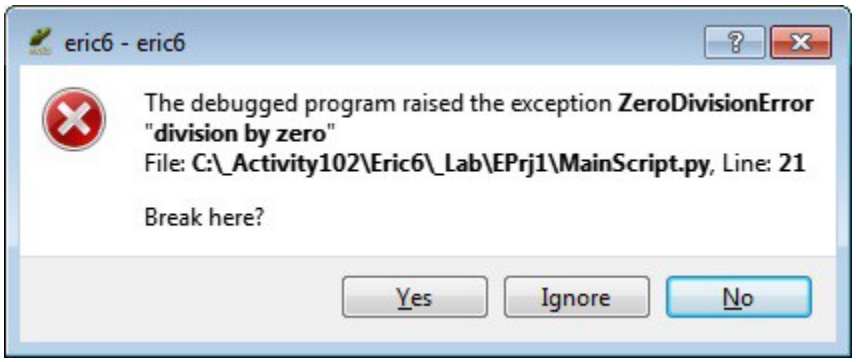
## Debug > **Ignored Exceptions...**

Designed to show a dialog box where to enter the code of which one of the Python handled exceptions[50] (such as: `TypeError`, `ZeroDivisionError`, …) are to be *ignored* during a debugging session, so that only all the others will be *considered*. This exception list is unique on Eric, and saved upon program exit. Possible Python un-handled exceptions remain unaffected.

Command aimed at focusing debug on chosen exceptions only [*see also:* `Exceptions Filter...`].

### *Remark*

Pressing the "`Ignore`" button on such a "`Break here?`" exception alarm box:

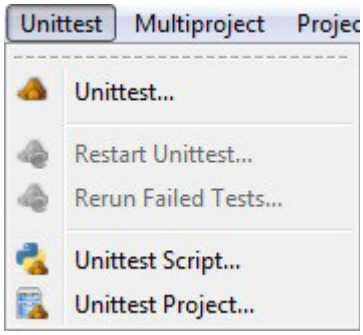



the involved exception code will be automatically added to this same `Ignored Exceptions` list [*see also: check box* `Report exceptions`*, at command* `Start > Debug`].

-<>-

49 That is, exceptions occurring within a "`try – except`" block.
50 That is, exceptions occurring within a "`try – except`" block.

# **Unittest** Command Menu

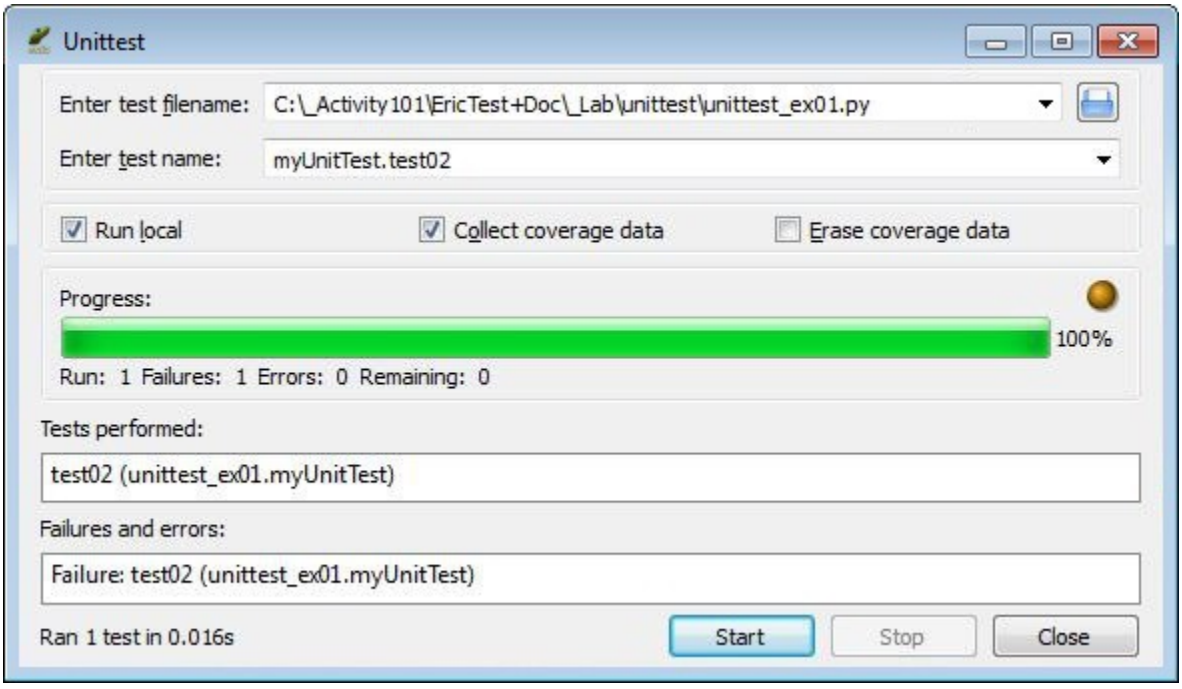



## *Command List*

```
unittest...          Restart unittest...  Rerun Failed Tests...
unittest Script...   unittest Project...
                                  - -
```

### Unittest > **`unittest...`**

Designed to show a "`Unittest`" dialog box to set and run a standard Python `unittest` procedure.



### *Specifications:*

`Enter test filename`
: "`*.py`" script file to perform the desired test, according to the Python "`unittest`" standard technique [*see:* Reference Book List, *in:* Appendix].

`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[51].

`Collect coverage data`
`Erase coverage data`
: To handle the "`Python Code Coverage`" data, as for `Start > Coverage Run of Script...` [*see*], related with these tests.

## *Remark*

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

– –

## Unittest > **Restart `unittest`...**

## Unittest > **Rerun Failed Tests...**

## Unittest > **`unittest` Script...**

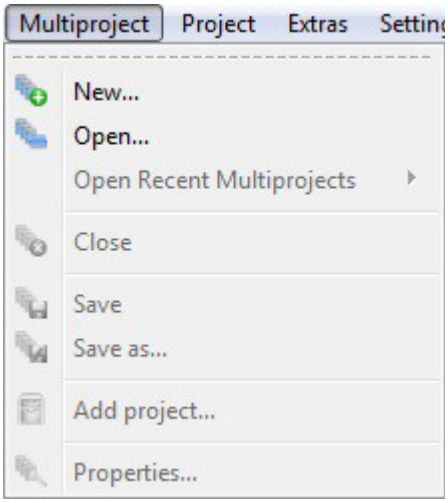
## Unittest > **`unittest` Project...**

Same as for the `unittest...` command [*see*], but with some form fields conveniently pre-filled with data derived from the current operative elements, such as the possibly opened script or Project.

–<>–

51 [*a feature we haven't tested*]

# **Multiproject** Command Menu




### *Command List*

```
New...      Open...          Open Recent Multiprojects     Close
Save        Save As...       Add Project...                Properties...
```

## *Remark*

As it's immediate to see a Project as a coherent collection of different Modules, it's likewise reasonable to see an Eric Multi-Project as a collection of different Projects which, for some reason, it's convenient to regard as a single macro-entity[52].

In a Multi-Project there is a special Main or Master Project, defined as the first to open. It's a concept corresponding to that of Main Script as defined in a Project [*cf.: next* Project Command Menu *section*] as the first source module to open and execute.

– –

52 Note that, as for Project, also Multi-project is a specific Eric concept, with no precise correspondence in Python terms.

Multiproject > **New...**

Multiproject > **Open...**

Multiproject > **Open Recent Multiprojects**

Multiproject > **Close**

Multiproject > **Save**

Multiproject > **Save As...**

Set of "`Multiproject`" commands corresponding to those aimed at handling Python scripts and Projects [*see menus:* `File >` *and* `Project >`], to which you may refer for detailed description. These the other commands hereafter described in detail, as specific of the Multi-Project feature.

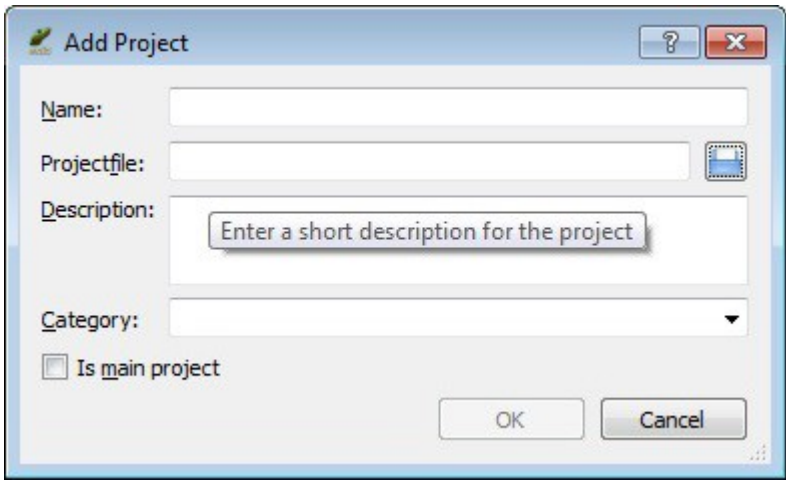
`Add Project...`

`Properties...`

– –

Multiproject > **Add Project...**

Designed to open a dialog box aimed at adding an existing Project to the currently opened Multi-project.

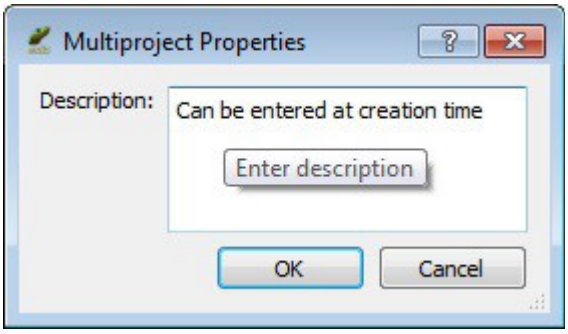



Dialog box assumed enough self-explanatory not to require any further description.

– –

## Multiproject > **Properties...**

Designed to display this "`Multiproject Properties`" dialog box, where a "`Description`" property can be first entered, then inspected.

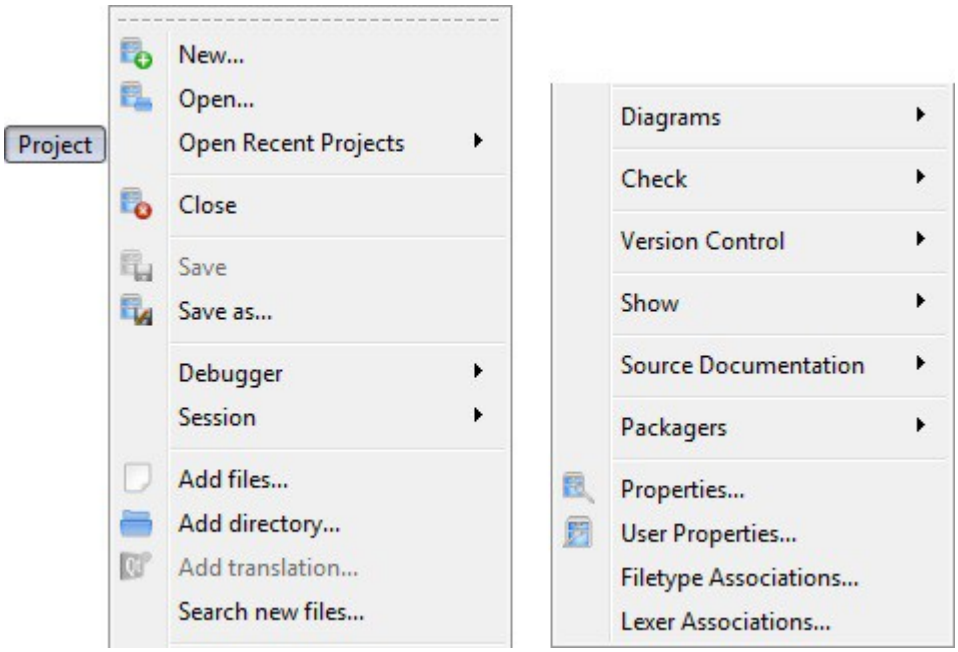



### *Remark*

Such "`Description`" property is here to be considered as the initial one of a possible set of properties, intended for future enhancements.

-<>-

# **Project** Command Menu




## *Command List*

```
New...        Open...       Open Recent Projects        Close     Save     Save As...
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...
```

– –

## *Remark*

<!> It's here worth recalling that it is always possible, if not even convenient, to treat even a single Python script as an Eric Project, so to enjoy the full set of commands and features as hereafter described. To that purpose, you just need to create a Project assigning to it the desired module as the "`Main Script`" [*see also next* Remark*, on this* Project Command Menu *section*].

An Eric Project, differently from a single Python script [*cf.: main menu* `File`], has got specific properties that can be both assigned at its creation and then also managed subsequently, with command menu `Project > Properties` [*see*].

<!> Some of these properties are worth to be recalled right away, hereafter:

`Project Directory`
: Project's host directory, in particular for[53]:

  `*.e4p` — An XML file to store the very Eric Project definition;

  `_eric6project` — A Project management sub-directory, automatically created to keep some complementary XML files.

## *Remark*

The default location for a new "`Project Directory`" is the so called Eric Workspace directory, initially set on the "`Select Workspace Directory`" control form at first Eric run [*see:* Setup and General Management, *in:* {6.Trail}]. The Eric Workspace can then be changed at any time on the "`Workspace`" field of: `Settings > Preferences... – Project > Multiproject, Configure multiproject settings` [*see*].

– –

`Main Script`[54]
: Project's entry script, where the execution will start. It's a concept corresponding to that of the "`Main Project`" in a Multi-Project [*see*].

`Version Control System`
: Property automatically asked at a new Project inception, concerning the possible adoption of a Version Control System (VCS)[55]—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.

<!> Note that 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 [*cf.:* `File > Close`]. That said also with reference to the re-synchronizing action required for debugging after the possible editing of a source script [*see*].

– –

53 <.> Note that the following identifiers are marked with two different ver. no.: 4 and 6; that is to say that not everything has changed with last Eric's ver. 6.

54 "Script" is a term not so infrequently used interchangeably with "Module" [*cf.:* Glossary, *in:* {Map}].

55 <!> Rather relevant a feature, well worth a Tech.Report of its own, as already done with "Eric 4 – Version Control" [*see*].

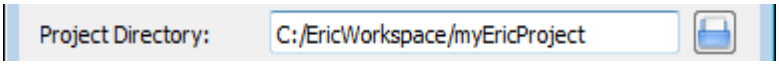
## Project > **New...**

A command corresponding to the analogous aimed at creating new Python scripts [*see in: menu* `File >`], to which you may refer for a detailed description.

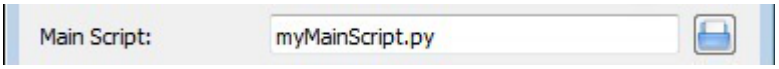
– –

### *Remark*

Then this `Project > New...` command will show the same "`Project Properties`" form that you'll find described in detail on the `Project > Properties...` command [*see*], just with a couple of fields here of particular interest.



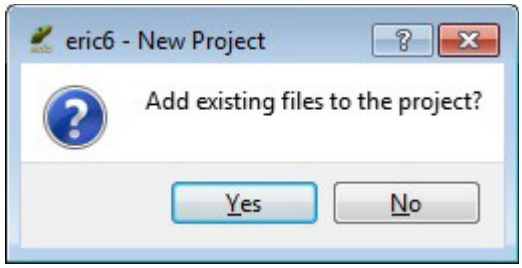

Possibly for a sub-directory into the root Eric Workspace, here offered as initial default[56].




Even not mandatory, as the Project Directory is, it's rather convenient here to preset a real Python module to be used as the initial main script.

– –

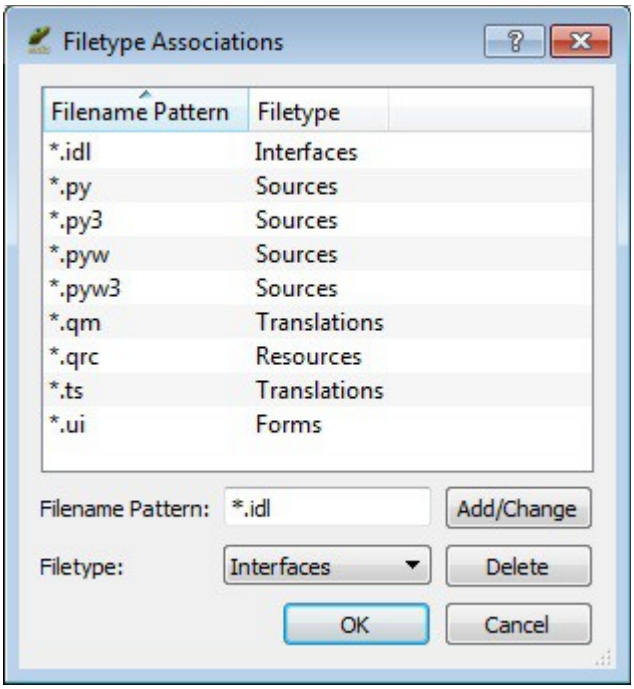
Not only an initial main script can be added to this new Project, but also any other file possibly already hosted into the selected Project Directory.




– –

56 <!> Note that the form's <OK> button will remain disabled until this initial default value remains unchanged.

Then, in case of actual addition of existing files, you'll be shown this other "`Filetype Associations`" form:




as better described on the related `Project > Filetype Associations...` command [*see*].

– –

## Project > **Open...**

## Project > **Open Recent Projects**

## Project > **Close**

## Project > **Save**

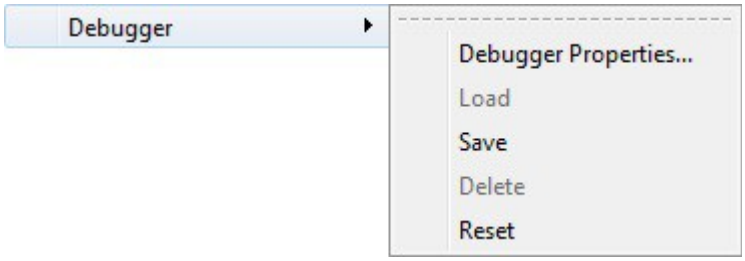
## Project > **Save As...**

Set of Project commands corresponding to those aimed at handling the Python script files [*see: command menu* `File >`], to which you may refer for detailed description. All the other "`Project >`" commands, as specific of the Eric Project structure, will be hereafter described in detail as usual.

– –

## Project > **Debugger**

Designed to call a sub-menu aimed at defining and managing the set of "Debugger Properties", here activated on a per-project basis.




This is a set of properties that, when defined, take priority over those defined in general with command `Settings > Preferences... – Debugger > …` [*see*], and that are stored into the current Project management directory "`_eric6project`", on a XML "*`myProject`*`.e4d`" file [*see*].

### *Sub-Command List*

`Debugger Properties...` `Load` `Save` `Delete` `Reset`

– –

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

Designed to show a dialog box where to possibly configure the specific Debugger Properties file for the current Project [*cf.*: *similar* `Settings > Preferences... – Debugger > …` *dialog boxes*].

### *Viewpoint*

<~> No detailed description of this feature will be here offered, as assumed off-scope for this Report [*cf.*: *section* Special Features, *in*: {0.Lead}].

– –

### Project > Debugger > **Load**

### Project > Debugger > **Save**

### Project > Debugger > **Delete**

### Project > Debugger > **Reset**

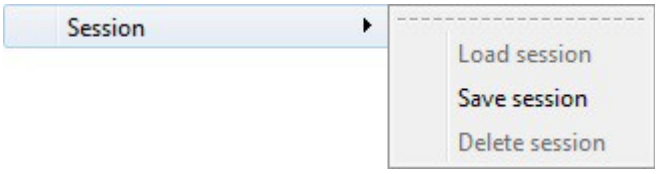
Suit of commands designed to manage the specific set of Debugger Properties possibly defined for the current Project. Note that these `Load` and `Delete` sub-commands are disabled when no such Properties file has been actually `Saved`.

– –

## Project > **Session**

Designed to call a sub-menu aimed at managing the set of operative parameters characterizing the current Eric's *Project Session*. That is, such working parameters as: opened source Text Forms, breakpoints, bookmarks, …

Actual behavior of this feature depends also upon the configuration of command: `Settings > Preferences... – Project > Project, Configure project settings, Sessions` [*see*].




Operatively a Project Session is such a XML text file: `<myProject>.e4s`, automatically named after the current Project and located into its standard management sub-directory `_eric6project` [*see*].

### *Viewpoint*

<!> A rather useful little feature, well worth knowing.

#### *Sub-Command List*

`Load Session` `Save Session` `Delete Session`

– –

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

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

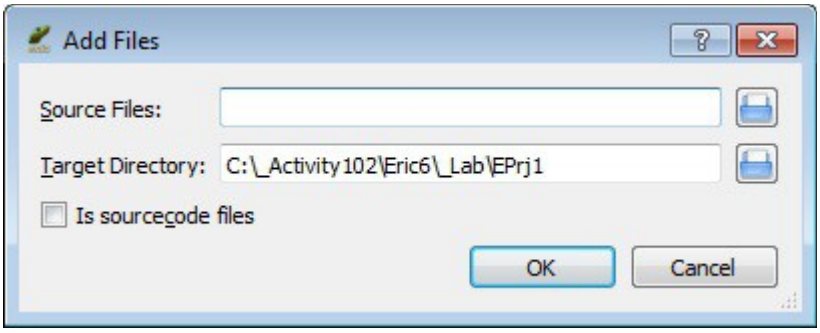
## Project > Session > **Delete Session**

Commands designed to actually manage—that is: activate, store, erase—the set of operative parameters characterizing the currently executing "Project Session". Note that these `Load` and `Delete` sub-commands are disabled when no such session file has been actually `Saved`.

– –

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

Designed to show a dialog box aimed at adding new files to the current Project.




### *Specifications:*

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

`Target Directory:`
Destination directory for the file(s) to be added to the Project

`Is sourcecode files`
To declare a file as source (in spite of any possibly different file extension)

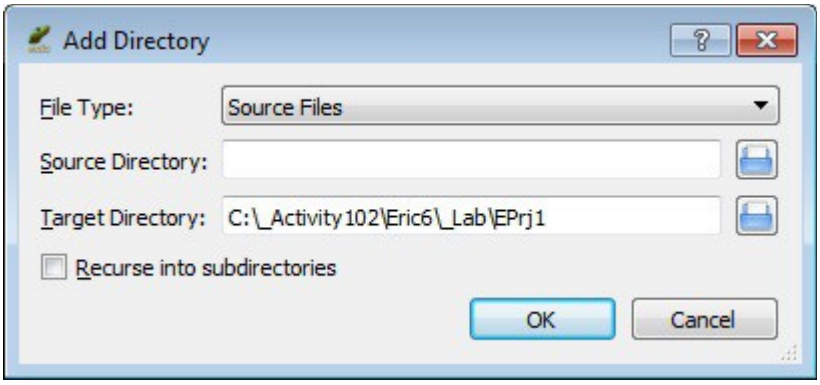
### *Remark*

<。> Files located in a directory other then the Target Directory will be physically copied, original files unaffected. If already there, this addition is purely "logical".

– –

## Project > **Add Directory...**

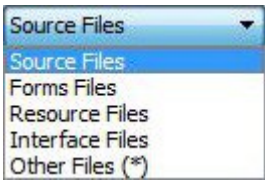
Designed to show a dialog box aimed at adding a whole directory of new files to the current Project.




An added directory will be then shown on the Project-Viewer form, there available to be managed with related context commands, `(^) Remove from Project` and `(^) Delete` included [*see in:* {4.West}].

### *Specifications:*

`File Type:` To possibly select the type of files to be added [*cf.:* `Project > Filetype Associations...`].




`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 logically added; or possibly also actually copied if it is different from the source directory. <!> Note that `Source` and `Target` directories may coincide, so to have it simply recognized as belonging to the Project and, therefore, searched in case of `Project > Search New Files...` [*see*].

`Recurse into subdirectories`
Addition action is to be extended also to the sub-directories' contents.

– –

## Project > **Add Translation...**

A command providing support for translating all the strings visible on the Graphic User Interface of the current Project into another given human language. A feature mainly based upon the Qt tools, and related to the `Translations Properties...` button of the "`Project Properties`" control form, as with commands: `Project > New...` and `Project > Properties...` [*see*].

Related features are also with commands: `Extras > Builtin Tools > Qt-Linguist...`, `> Translations Previewer...`, and the context menu of *Project-Viewer:* Translations on the L-Pane [*see in:* {4.West}].

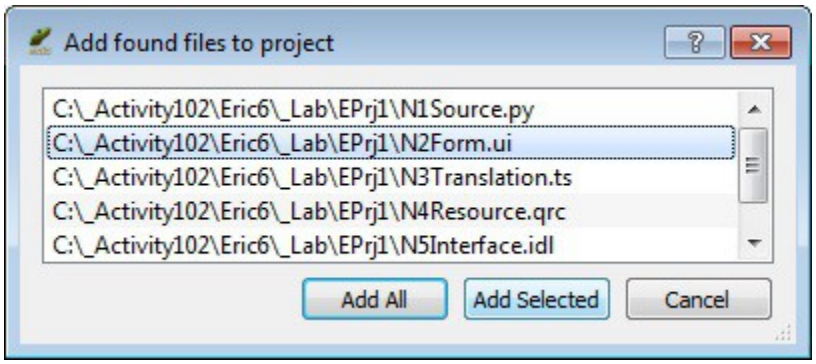
### *Viewpoint*

No further detailed description of this feature will be here offered, as assumed off-scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

– –

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

Designed to display the "`Add found files to project`" dialog box aimed at possibly adding to the current Project some of the files found in the Project's directory and not yet belonging to it.

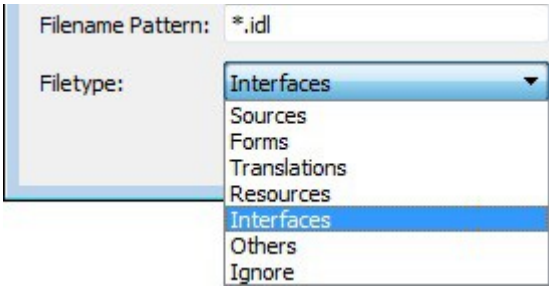



The directory searched is the "`Project Directory`", as set on the "`Project Properties`" form [*see: command* `Project > Properties...`], with the search recursively extended to the Project's sub-directories only if known to Eric, otherwise ignored [*cf.:* `Project > Add Directory...`].

<。> Added files will be then shown on the Project-Viewer form, where they can be managed by means of the related context commands, notably `(^) Remove from Project` and `(^) Delete`, to perform an action inverse to this add [*see in section* {4.West}].

### *Remark*

In this "`Add … Files`" form you'll see conveniently listed only the new files assumed of interest for the current Eric Project, as declared with command `Project > Filetype Associations...` [*see*].




That is: `Sources`, `Forms`, `Translations`, `Resources` and `Interfaces` files; with the exclusion of the Filetypes: `Others` and `Ignore`.

– –

## Project > **Diagrams**

Designed to call a sub-menu of commands aimed at creating a symbolic diagram, that is a graph-based schematic representation of the Python Application under development in the current Project.


### *Viewpoint*

A self-standing subject of specific relevance, here just hinted at, as assumed off-scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

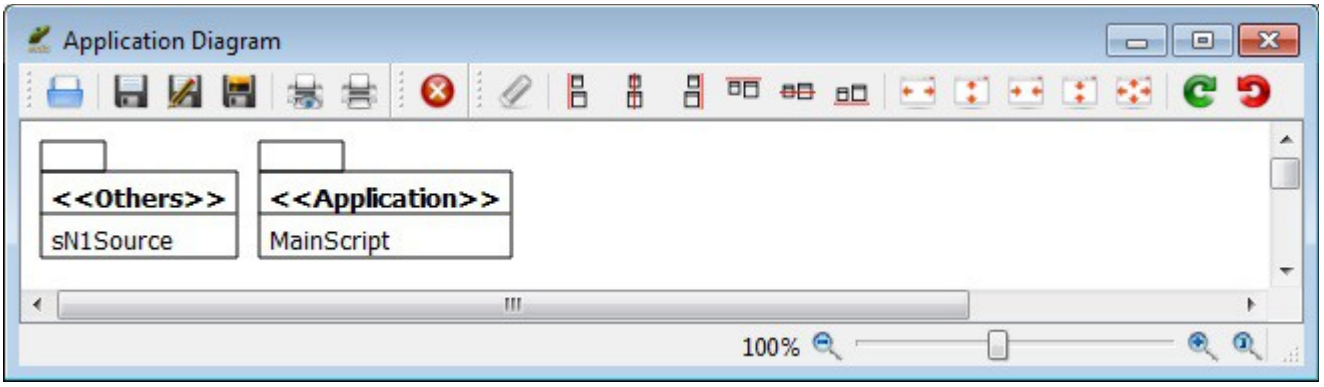
### *Sub-Command List*

`Application Diagram...` `Load Diagram...`

– –

## Project > Diagrams > **Application Diagram...**

Designed to open the "`Application Diagram`" graphic editor, possibly initialized, upon user's request, with the current Project's module names.

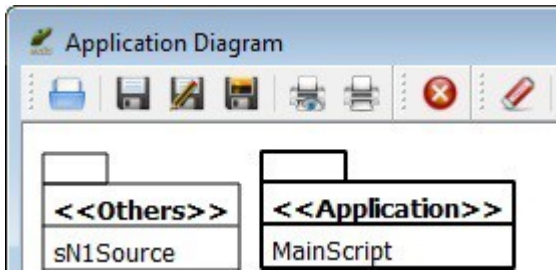



The resulting diagram can be saved as an "Eric Graphics file", of type "`*.e5g`", to be then possibly re-opened and further edited [*see: next* `Diagrams > Load Diagram...`].

– –

## Project > Diagrams > **Load Diagram...**

To load and open an "Eric Graphic file", of type "`*.e5g`", so to be then possibly further edited[57] [*cf.: former* `Diagrams > Application Diagram...`].

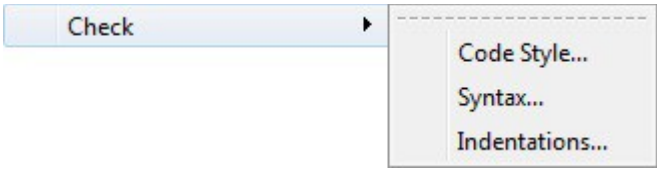



-<>-

57 <~> Here, as initial default `Load` directory, we'd suggest a better choice: the same Project directory, as with the corresponding `Save` command [*see*], instead of the Python's installation directory.

## Project > **Check**

Designed to call a sub-menu of commands aimed at performing some standard source checks on the current Project.




### *Viewpoint*

A subject here just hinted at, as assumed off-scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

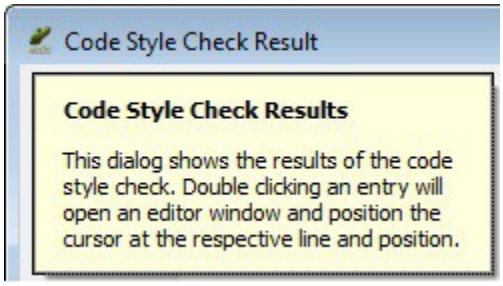
### *Sub-Command List*

`Code Style...` `Syntax...` `Indentations...`

– –

## Project > Check > **Code Style...**

[58]To quickly scan the current Python Project for compliance with such Python coding guidelines concerning the naming conventions, source code documentation and the so called "PEP 8" style guide[59].




58 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].

59 Python Enhancement  Such "`Description`" property is here to be considered as the initial one of a possible set of properties, intended for future enhancements. Proposals (PEPs), PEP 8 -- "Style Guide for Python Code", 05-Jul-2001.

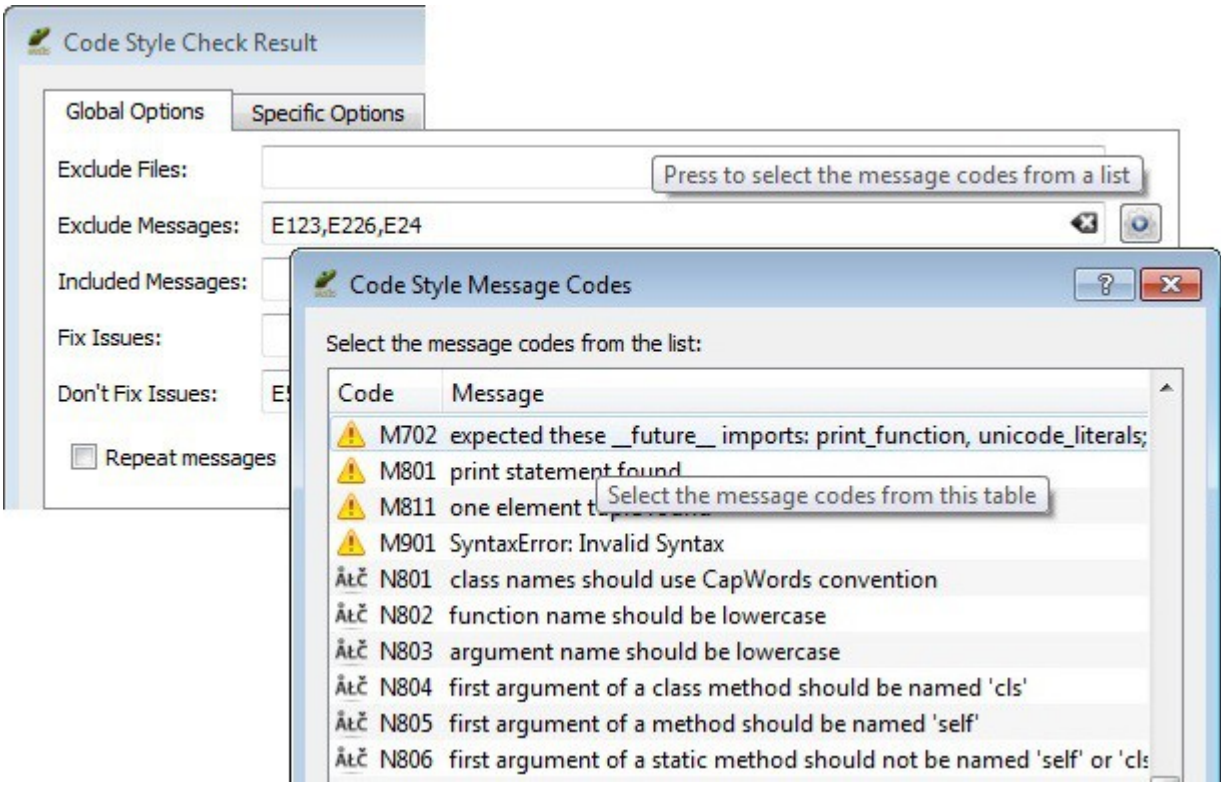
The dedicated "`Code Style Check Result`" form is for the management of this check, and for the display of consequent results [*see*].

## *Remark*

A dedicated "`noqa`" (no-Quality Assurance) Editor flag[60], such as:

```
#eflag: noqa = M601,M702
```

is available to be positioned as an encoding-comment line *at the end* of any source module file, so to possibly suppress messages generated by this command. Such flag expects a comma separated list of message codes as those shown on this "`Code Style Message Codes`" snapshot.




## *Viewpoint*

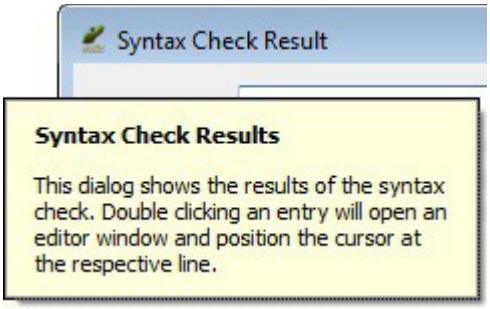
<~> No further detailed description of this rather relevant command feature will be here offered, as assumed off-scope for this Report [*cf.: section* Special Features*, in:* {0.Lead}].

-<>-

60 Cf. also the "`#eflag: FileType = …`" Editor flag, at How to Select the Language Interpreter, in: {0.Lead}.

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

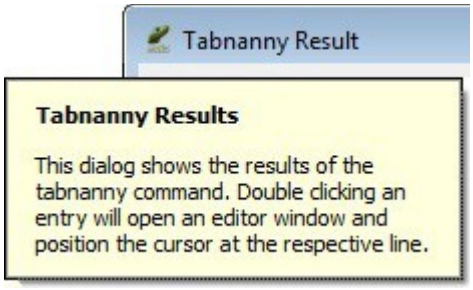
[61]To quickly scan current Python Project for syntax errors[62], possibly displayed on a "`Syntax Check Result`" window.




– –

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

[63]To quickly scan current Python Project for indentation errors, possibly displayed on a "`Tabnanny Result`" window.




### *Remark*

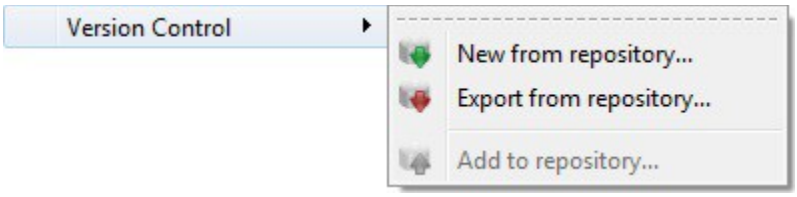
"*`Tabnanny`*" is a standard Python service module, aimed at the "*Detection of ambiguous indentation*". But, having to do with semantics, be warned that this test is not, and couldn't possibly be, a sure bet.

– –

61 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].
62 Making use of "PyFlakes", the popular checker of Python programs.
63 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].

## Project > **Version Control**

[64]Designed to show a sub-menu of commands aimed at managing the Version Control System (VCS)—i.e.: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted for the current Project.




### *Viewpoint*

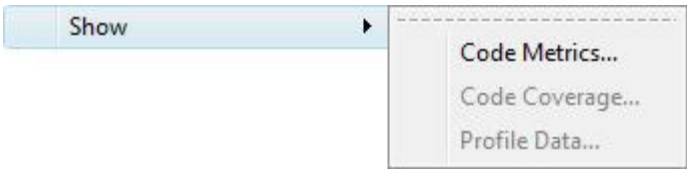
<。> This is a self-standing subject of particular relevance, here just hinted at as assumed off the scope of this Report [*see:* Scope of this Report, *in:* {0.Lead}].

In fact, this is a function deserving a dedicated treatment as already done with the "Eric 4 Version Control" Technical Report, to which you may refer[65] as it is basically valid for this Eric version too.

– –

## Project > **Show**

Designed to call a sub-menu of commands aimed at computing and displaying some standard statistic data about the current Python Project.




### *Sub-Command List*

`Code Metrics...` `Code Coverage...` `Profile Data...`

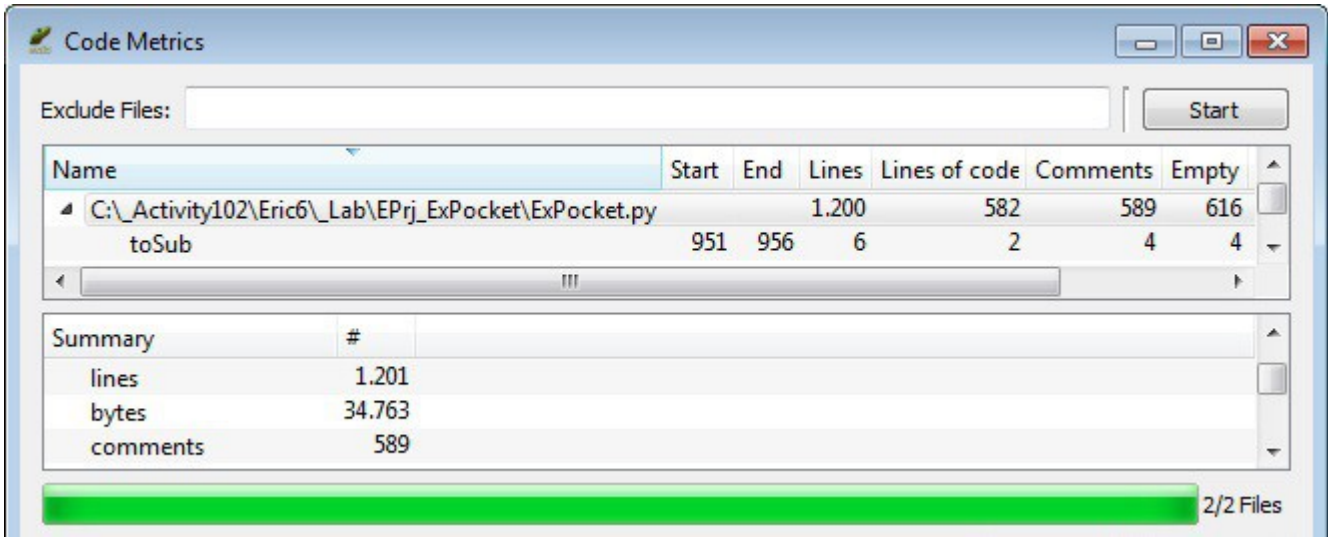
– –

64 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].

65 See related URL in: Foreword, *What & Where in the Internet*

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

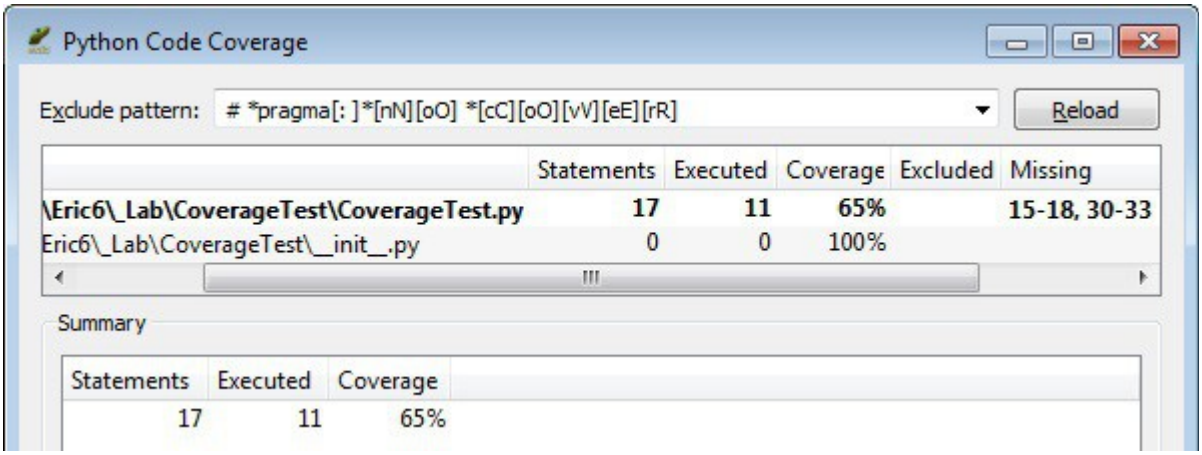
Designed to show a "`Code Metrics`" window, where to display a comprehensive set of "static" census data about the source code; such as: name and number of files, count of empty lines, …




– –

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

Designed to show a "`Python Code Coverage`" window, where to display the comprehensive set of "dynamic" census data possibly collected and stored during last "`Start > Coverage Run`" session [*see*]. Command disabled if no prior "`Coverage`" run executed.



Where, in particular:

`Exclude pattern`
: Regular expression pattern, as described calling the related `What's This?` help [*see*];

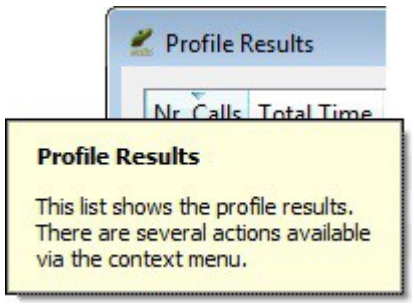
`Reload`
: Button useful in case of an "`Exclude pattern`" change;

`Summary`
: Area relevant in case of coverage analysis of a Project, not in case of a single module.

– –

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

Designed to show a "`Profile Results`" window, where to display the comprehensive set of "dynamic" census data possibly collected and stored during last "`Start > Profile Project...`" session [*see*].

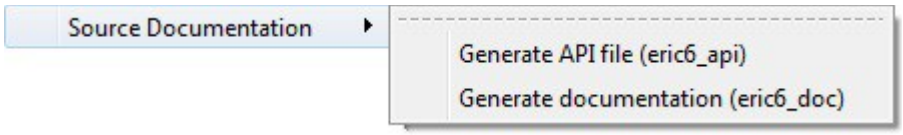



Disabled if no "`Profile`" run executed.

– –

## Project > **Source Documentation**

Designed to call a sub-menu of commands aimed at generating "`API`" and "`Documentation`" files derived from the current Eric source Project.



### *Remark*

The two commands shown in this sub-menu correspond to the "`Ericapi`" and "`Ericdoc`" items listed on the `Plugins > Plugin Infos...`'s "`Loaded Plugins`" form [*see*], where they are also briefly described; and, as Python scripts, they are stored into such Eric's standard setup repository as: `...\Plugins` [*see*]. A structurally similar collection of sub-commands can be found at `Extras > Wizards` [*see*].

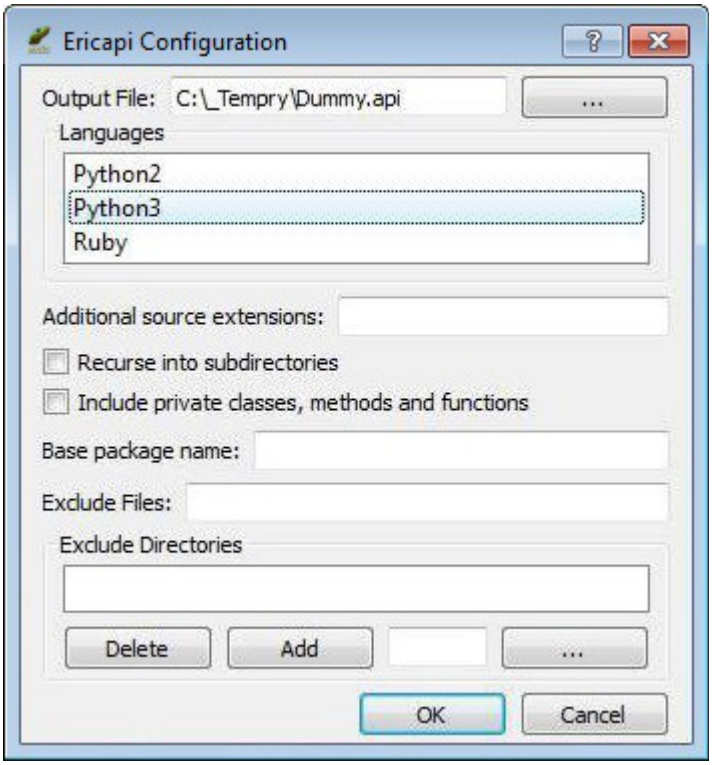
### *Sub-Command List*

`Generate API File (eric6_api)` `Generate Documentation (eric6_doc)`

– –

## Project > Source Documentation > **Generate API File (eric6_api)**

[66]Designed to show such an "`Ericapi Configuration`" control box:




66 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].

meant to set up the desired parameters for the "`eric6_api`" script, and then to execute it with the purpose of scanning Python (/ Ruby) source files so to extract information that can be used later on, by auto-completion and call-tips functions [*cf.:* `Edit > Complete` *sub-commands*].

– –

### *Remark*

This very "`eric6_api`" script is part of a standard Eric installation and can be run also autonomously, independently from Eric [*see: among the "*`*.bat`*" files, in:* `C:\Python3X` *directory; after a standard Eric installation*], and will generate "`Output File`" of "`*.api`" type, which are text files with a contents of this kind:

```
PythonCad._initialize_styles?5()
PythonCad._inizialize_snap?5()
PythonCad.main?4()
```

– –

### *Viewpoint*

A subject connected with the `Edit > Complete` and `Edit > Calltip` commands [*see*], here just hinted at as assumed off-scope for this Report [*see:* Scope of this Report*, in:* {0.Lead}].

– –

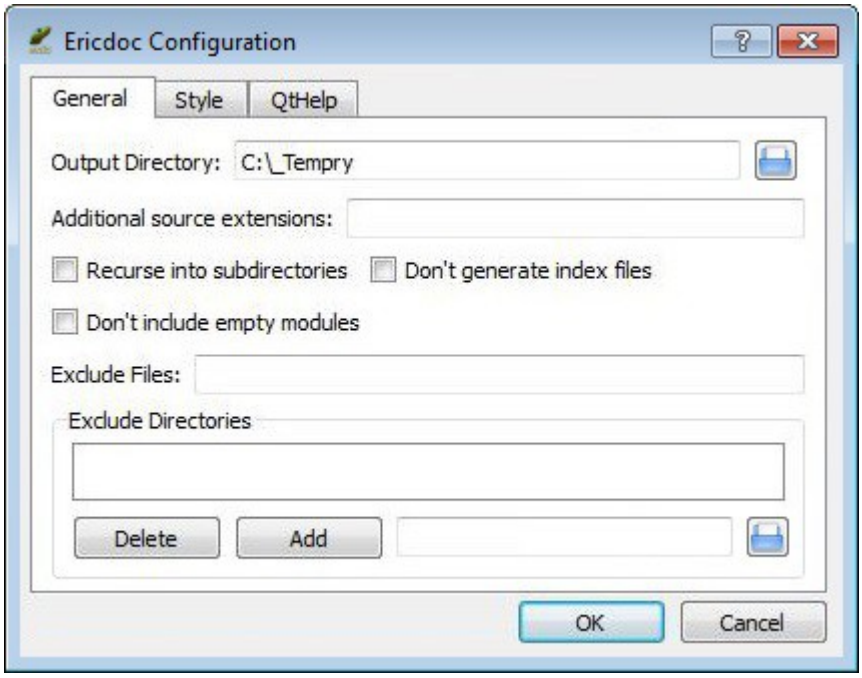
## Project > Source Documentation > **Generate Documentation (eric6_doc)**

[67]Designed to show such an "`Ericdoc Configuration`" control box, aimed at managing the generation of a set of documentation files derived automatically from all source items contained into the current Eric Project's host directory[68].

That is, such source items as: Packages, Modules, Functions, Classes, Attributes, Methods, …; possibly being commented with the related standard Python "`Documentation Strings`", when present on the source text. Note that these are precisely the same content items as shown on the "`Source`" tab form of the the `Project-Viewer` [*see:* Application Window Map *to locate the* `Project-Viewer`*, in:* {Map}].

67 Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].

68 Which implies that, beside the very Project, also all other items possibly there located will be processed.
<。> Use the "`Exclude`" option to ignore unwanted items.




The resulting documentation is recorded into "`*.html`" files, with names reflecting those of the corresponding items. A set of documentation files well suited to be browsed via `Help > Helpviewer...` [*see*].

– –

## *Remark*

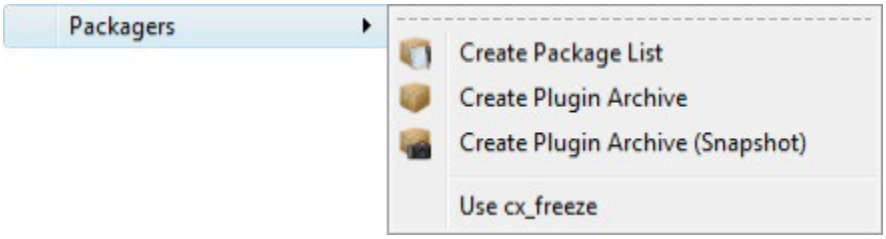
Controls on this dialog box are assumed as clear enough not to require any further description. <。> Just a suggestion: Set the check-box "`Recurse into subdirectories`" if you want to scan all of your Project's directory tree.

Then, of course, the resulting quality of the documentation generated this way will necessarily reflect the quality of the Python standard "`Documentation Strings`", as actually inserted on top of each source item of the Project.

-<>-

## Project > **Packagers**

Designed to call a sub-menu of commands aimed at the development of Eric "*Plugin*" add-on tools [*see: next section* Plugins Command Menu*, and command:* `Extras > Plugin Tools`].




This sub-menu is normally seen disabled, being enabled only when working on an "`Eric Plugin`" Project Type, as can be set on the `Project > Properties... – Project Properties` control box [*see*].

– –

### *Remark*

<!> This whole subject is not related with the ordinary use of Eric, but with the development of original Eric Plugin add-ons instead. As such, it is assumed as a self-standing subject of particular relevance, here just hinted at as assumed off-scope for the current Report [*see:* Scope of this Report*, in:* {0.Lead}].

### *Sub-Command List*

`Create Package List` `Create Plugin Archive` `Create Plugin Archive (Snapshot)`
`Use cxfreeze`[69] [*optional, see:* `Plugins > Install Plugins...`]

– –

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

Designed to create a list of files to be included into a Plugin Archive.

– –

69 <~> To be precise, "`cxfreeze`" is the name of the single script as currently available with Eric, whereas "`cx_Freeze`" is how it is usually named the entire set of scripts comprising this product.

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

Designed to create a Plugin Archive for a release version. Plugin Archives can be added to the Eric plugin system by means of the `Plugins > Install Plugins...` command [*see*].

– –

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

Designed to create a Plugin Archive for a development version (i.e. a so called "snapshot").

– –

### Project > Packagers > **Use `cxfreeze`**

Here present just as a notable example of this `Project > Packagers` sub-menu extended with a standard Pluging package [*see:* `Plugins > Install Plugins...`].

– –

### *Viewpoint*

<!> What here follows is rather a remarkable point, well worth knowing.

Note that this is not an Eric standard command but, as already declared [*cf. also:* `Plugins > Install Plugins...`], an optional add-on operating as a calling frame for the well known cx_Freeze utility tool; provided this last has been independently installed on the very system in use.

Declared purpose of such a tool is: "*freezing Python scripts into executables*", as reads its web site [*see: URL* `http://cx-freeze.sourceforge.net/`]. That is, nothing less than a tool for the natural conclusion of a s/w designing process, that is: distribution.

– –

The distribution of Python s/w products, even though considered exclusively by means of cx_Freeze, is a topic certainly exceeding the purpose of this Report [*see:* Scope of this Report*, in:* {0.Lead}], but also so relevant that we've carried on a dedicated investigation, and produced two Reports[70] concerning the subject:

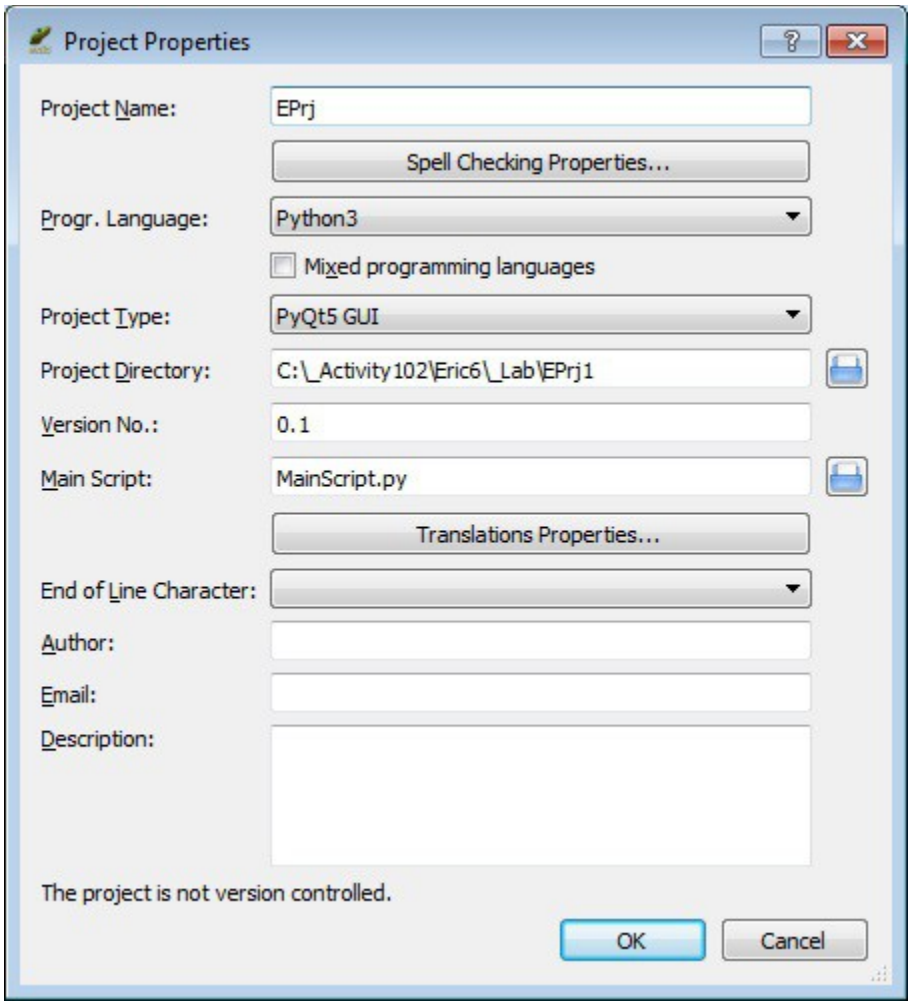
- One about the limited "`cxfreeze`" technique, as currently available with Eric, and as here hinted at;
- And one about the more powerful cx_Freeze "`distutils`" technique, here envisioned as a possible considerable enhancement for Eric.

-<>-

70 <~> Both unpublished, for lack of confidence on the subject. If anyhow interested, try contact the Author.

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

Designed to show a "`Project Properties`" form, where to manage current Project's properties. Note that it's exactly the same form shown at the creation of a new Project [*cf.:* `Project > New`].



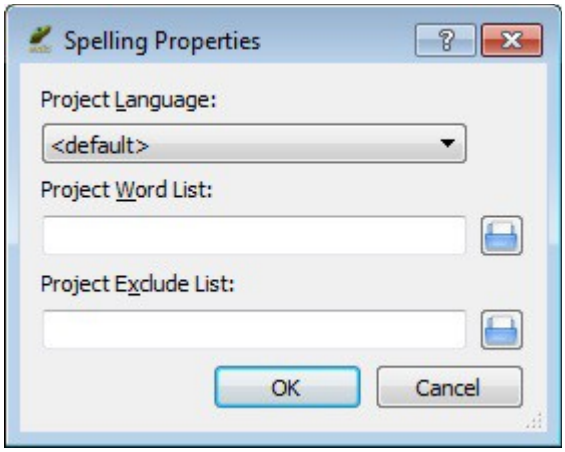

### *Specifications:*

`Project Name` The original Project name, as assigned at the `Project > New...` creation time [*see*], can be here changed in this rather particular way:

A new "`*.e4p`" Eric Project file will be created, with the new name. The old version of the same file is kept unchanged, and could be then possibly eliminated by hand.

`Spell Checking Properties...`

Button to show a "`Spelling Properties`" control box, where to possibly define on a per-Project local scope some of the basic properties as set globally, per-User, via the `Settings` > `Preferences... - Editor > Spell checking` command; and taking precedence over them [*see*].




These settings are related to the spell checking feature, as with the "`Extras >`" main menu commands and with the text edit context "`Spell (^)`" menu commands [*see*].

Language choice: `de`, `en` (flavors: `US`, `GB`, `AU`), `fr`; with a "`<default>`" as set in the above cited "`Spell checking`" global settings [*see*]. Default storage location is here the same Project's host directory.

## *Remark*

<.> Spell checking is an optional feature that requires the presence of the PyEnchant toolkit [*see:* Optional Packages, *in:* {6.Trail}*; cf. also:* `Settings > Show External Tools` *list*].

– –

`Progr. Language`

A drop-down list offering a choice amongst all possibly available programming languages, such as Python 2, Python 3 and Ruby. That is, beyond the very Python version in use as prerequisite [*see:* Compatibility with Python ver. 3 or/and 2, *in:* {0.Lead}, *and also* `Shell (^) Start` *context command, in:* {3.South}].

`Mixed programming languages`

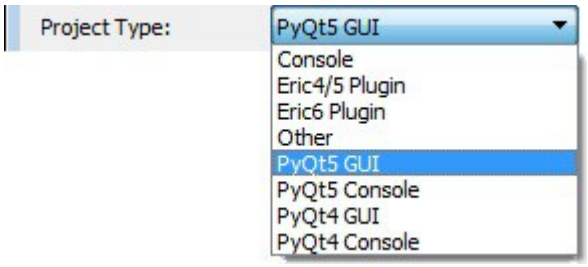
A check-box for accepting as valid Project modules all file types associated with any of the known languages – otherwise the only one associated with the selected `Progr. Language` parameter [*see*] will be accepted.

## *Remark*

It's here worth recalling that the Eric IDE is primarily aimed at the Python language, but can also somehow be used with the language Ruby [*see:* Eric's Compatibilities, *in:* {0.Lead}]. A feature, anyhow, here just hinted at, and not specifically treated in this Report.

– –

`Project Type` A pre-established list of choices:




## *Remark*

A Project Type list that can be field-extended with other so called Project Plug-ins, such as the Django type, as available in the Internet via `Plugins > Plugin Repository...` [*see, and see also:* `Plugins > Install Plugins...`].

– –

`Project Directory`
Root "Workspace" directory for the current Project, initially set at the Project creation [*see also: former section* Project Command Menu, *first* Remark]

`Version No.` Usual Python meta-data

`Main Script` Project's entry script, where the execution will start

`Translations Properties...`
To show the property dialog box related with the command `Project > Add Translation...` [*see*].

## *Viewpoint*

<~> About this "`Translations`" matter we'd welcome some specific remarks from experienced users, so to go beyond this rather generic description, of limited practical use.

– –

`End of Line Character`
: To chose an End-Of-Line type possibly different from that one preset in `Settings` > `Preferences... - Editor` > `Filehandling`, `End of Line Characters` [*see also:* `Edit > Convert Line End Characters`].

`Author, Email, Description`
: Usual Python meta-data

– –

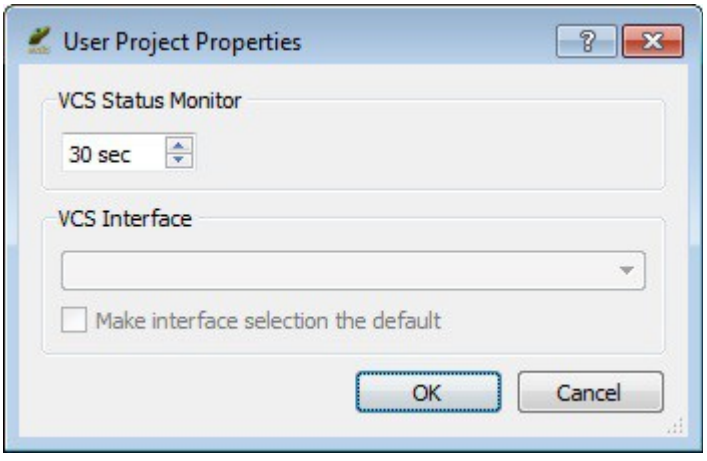
### *Remark*

<!> Here such bottom-caption: "`The project is not version controlled`" [*see*] is related to the Version Control System (VCS)—i.e: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted, as with menu command `Project > Version Control` [*see*].

– –

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

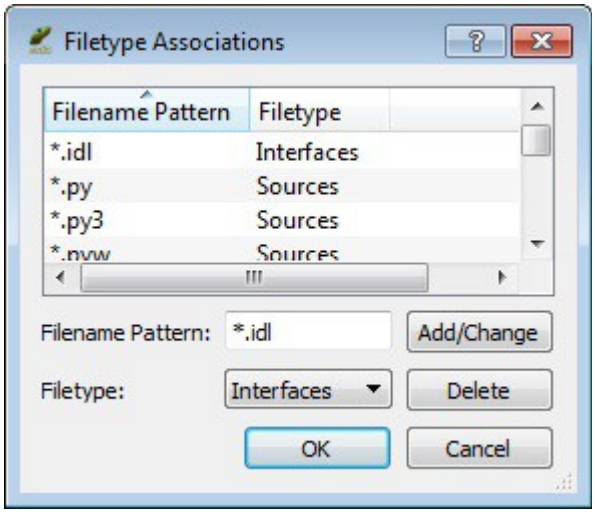
Designed to show a "`User Project Properties`" box, just aimed at displaying some info about the Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted for the current Project [*see: command* `Project > Version Control`].




– –

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

Designed to show a dialog box where to see and manage which the file extension is to be associated to the different files normally used in a Project.



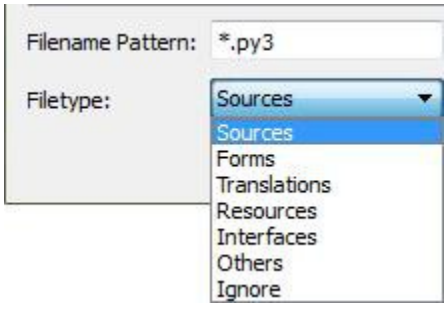

More precisely this command can associate file-name "Patterns"[71], a concept including such sheer "`*.py`" file extension as a particularly common and simple case.

<。> This is a defaulted Project-level setup, with impact on such commands as: `Project > New...`, `> Add Directory...` and `> Search New Files...` [*see*].

– –

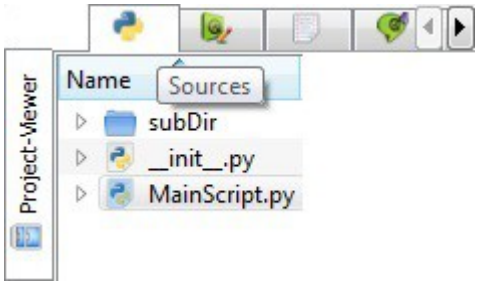
### *Remark*

Note here the "`Filetype`" drop-down list, showing which are considered the most common Eric's file-types in use.




71 Also known as: "wildcard pattern".

<。> Note also the precise correspondence between this `Filetype` list and the `Project-Viewer` tabs: `Sources`, `Forms`, `Resources`, `Translations`, `Interfaces (IDL)`, `Others` [*see: section* ***L-Pane***: Project-Viewer, *in:* {4.West}];

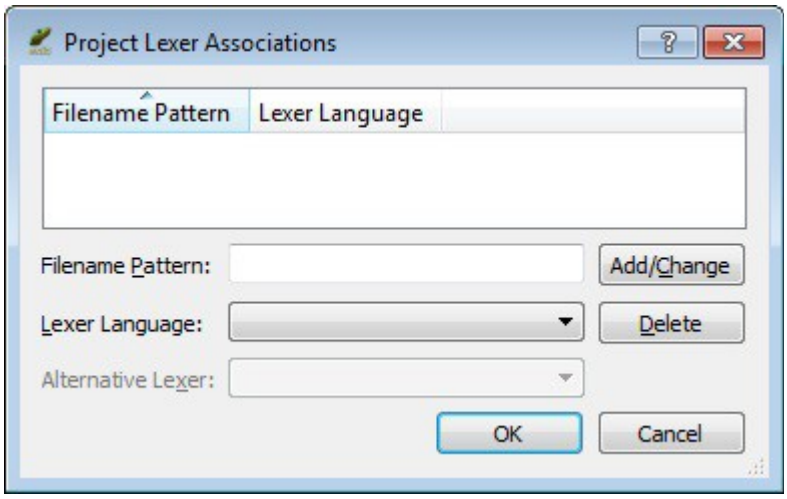



with particular reference to the special types "`Others`", all shown under the same tab, and "`Ignore`", there not shown at all.

– –

## Project > **Lexer Associations...**

Designed to show a "`Project Lexer Associations`" box where to establish the correspondence between source files and coding languages. Correspondence that will be used by the Eric's lexical analyzer tool[72]—"lexer", for short—to conveniently highlight and "colourize" the text shown on the Eric's Text Edit Pane, so to enhance readability [*cf.: first* ***Remark*** *at the* Text Edit Central Pane, *in:* {2.Central}].




72 <。> Actually, two [*see: next* ***Specifications***]: QScintilla, as included in PyQt [*cf.:* Prerequisites, *in:* {6.Trail}], or Pygments syntax highlighter [*cf.: command* `Settings > Show External Tools`].

A per-Project custom setting, provided to possibly take precedence over the corresponding global correspondence as found in: `Settings` > `Preferences...` – `Editor` > `Highlighters` > `Filetype Associations` [*see*], as factory pre-set.

### *Specifications:*

`Filename Pattern`
: Also known as "wildcard pattern" that is, in its simplest form, a "`*.xyz`" file type extension.

`Lexer Language`
: Associated language among the (many!) recognized by the QScintilla lexical analyzer here listed; unless the special "`Alternative`" item is chosen, so to switch the choice to the next drop-down list [*see*].

`Alternative Lexer`
: Extra drop-down list of the (many!) other languages recognized and handled by the Pygments lexical analyzer, enabled when the "`Alternative`" item is chosen at the bottom of the former "`Lexer Language`" drop-down list [*see, and cf.:* `Alternatives` *sub-command of* `Text Form (^) Languages`, *in:* {2.Central}].

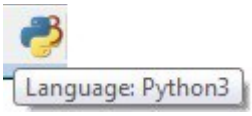
– –

## *Remark*

Other features someway related with this same command are:

`Settings` > `Preferences...` – `Editor` > `Highlighters` > `Filetype Associations`
: For the same "`Associations`" but, as said, valid at the global level.

`Text Form (^) Languages`
: A context command, taking local (per-Edit) precedence over this per-Project command [*see in:* {2.Central}].



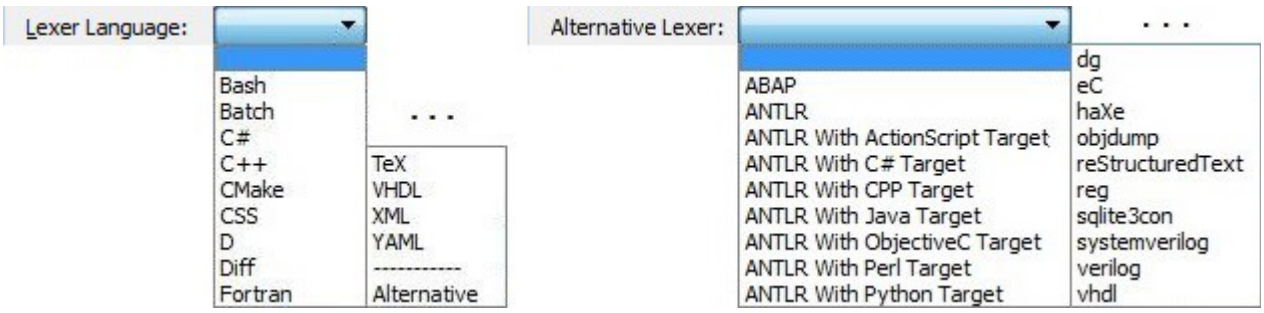

Syntax Highlighter symbol, provided to tell which language is currently and locally in use [*see:* Information Bar, *in* {3.South}].

*Shebang* line, e.g.: `#! [...] PythonN`
: Be aware that, being this "`Lexer Associations`" based upon the source Filename Pattern, a possibly different language choice made via such esoteric way as the Shebang line [*see:* How to Select the Language Interpreter, *in:* {0.Lead}] will be here simply ignored.

– –

## *Remark*

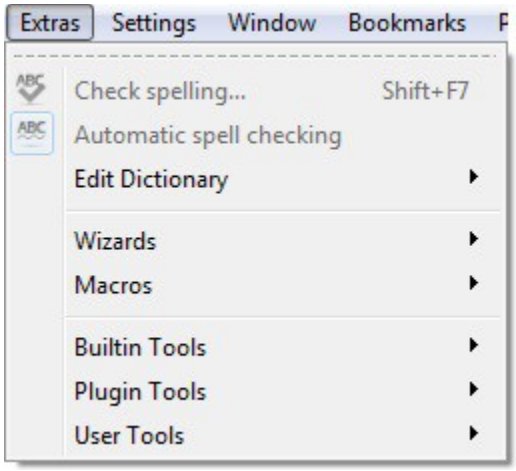
In case you'd wander about the reason for such an impressive Lexer Languages drop-down lists, be reassured:




they have very little to do with Eric, being simply the lists of all languages for which the lexers here in use have got a highlighting pattern. Among which will remain of interest for you only the few you'll be actually dealing with when working on this IDE.

-<>-

# **Extras** Command Menu




### *Command List*

`Check Spelling...` `Automatic Spell Checking` `Edit Dictionary` `Wizards`
`Macros` `Builtin Tools` `Plugin Tools` `User Tools`

– –

## Extras > **Check Spelling...**

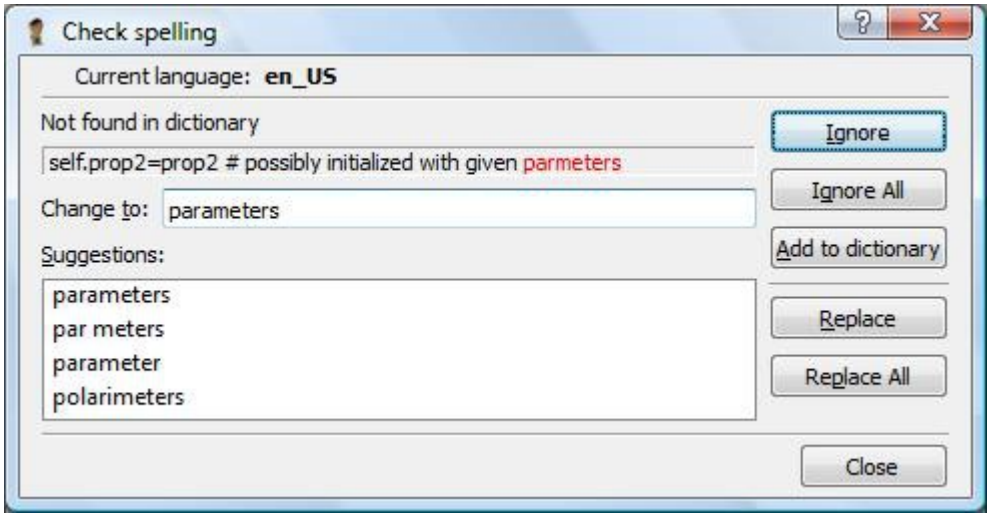
Designed to show a "`Check spelling`" control box where to check and manage the possible spelling issues detected in the text written on the current Text Edit Form; typically comments.

– –

### *Remark*

<。> This command requires the presence of PyEnchant, optional spell-checker toolkit [*see:* Optional Packages*, in:* {6.Trail}*; cf. also:* `Settings > Show External Tools` *list*]. Being such toolkit not installed, no check spelling feature is available, and this command results "dim"-disabled.

– –




### *Specifications:*

`Current Language`

Displays current choice among: `de`, `en` (flavors: `US`, `GB`, `AU`), `fr`; as set via global `Settings` or local `Project` properties [*see next* ***Remark***].

`Add to dictionary`

Button provided to possibly add to the Dictionary in use a word here declared as "`Not found in dictionary`" [*cf. also: context* Check Spell Menu*, in* {2.Central}]. Note that there are two of such Dictionary sets, one global per-User and one local per-Project, with the latter taking precedence over the former. This button is meant to add to the global per-User one [*see next* ***Remark***].

`Ignore` / `Replace`

Actions meant to act on the current edit session only, if no "`Add to dictionary`".

## *Remark*

Global spell-check options—notably: Language and Dictionary[73]—can be managed with command `Settings > Preferences... – Editor > Spell checking` [*see*]. Where, in case of PyEnchant not installed, you will be informed that "`Spell checking with PyEnchant is not available`". Default storage location: "`<User>\_eric6`" directory; being this a personal, i.e. per-User —that is: *not* per-Eric installation—, global setting.

Language and Dictionary can be also defined at a per-Project level, where it will take precedence, with command `Project > Properties... – Spell Checking Properties...` [*see*]. Default storage location: Project's host directory.

73 Actually, a set of two files: a "Word list", and an "Exclude list".

<。> Contents of all these Dictionaries can be managed with the next "`Extras > Edit Dictionary`" commands [*see*]. But still, being them plain text files, they can also be examined and, possibly, "bulk"–edited with an ordinary text editor.

– –

## Extras > **Automatic Spell Checking**

Designed to enable / disable the automatic spell checking of the text on the currently active Text Edit Form. Possibly mis-spelt words are merely signaled with a red wavy underlining [*see snapshot*], otherwise unchanged.

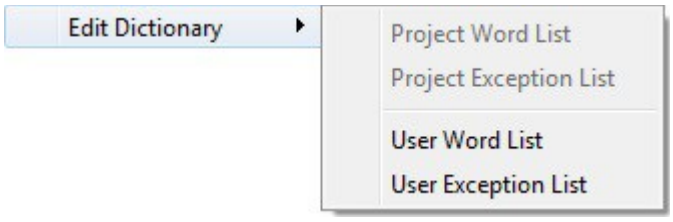

### *Remark*

<。> Also this command requires the presence of PyEnchant, optional spell-checker toolkit. For actual spell editing features see above command: `Extras > Check Spelling...`

– –

## Extras > **Edit Dictionary**

Designed to call a sub-menu of commands aimed at managing the per-Project (local) and per-User (global) spell-check dictionaries.




### *Sub-Command List*

`Project Word List` `Project Exception List` `User Word List` `User Exception List`

– –

Extras > Edit Dictionary > **Project Word List**

Extras > Edit Dictionary > **Project Exception List**

Extras > Edit Dictionary > **User Word List**

Extras > Edit Dictionary > **User Exception List**

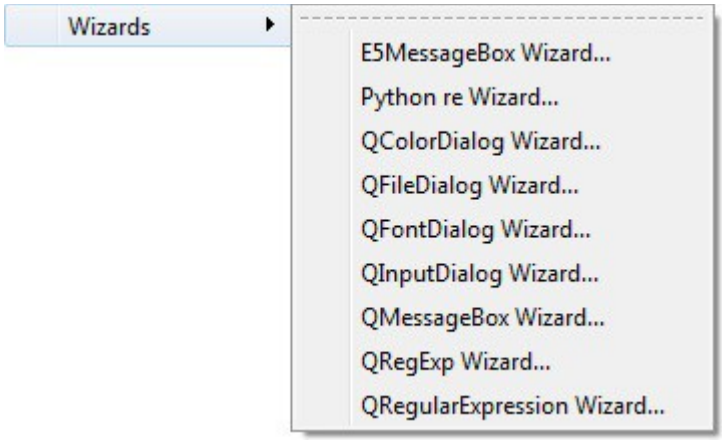
Set of sub-commands designed for editing the named dictionaries, disabled in case of a dictionary not defined. Commands assumed self-explanatory enough not to require any further description.

About the creation, location and use of such dictionaries, refer to the above command: `Extras > Check Spelling...` [*see*].

-<>-

## Extras > **Wizards**

[74]Designed to call a sub-menu of commands aimed at running a coherent set of Plugin "wizards", here conveniently collected together.




### *Remark*

The commands shown in this sub-menu correspond to last nine "`Wizard`" items listed on the `Plugins > Plugin Infos...`'s "`Loaded Plugins`" form [*see*], where they are also briefly described. And, as Python scripts, they are stored into such Eric's standard setup repository as: `...\Plugins\WizardPlugins` [*see*]. A structurally similar collection of sub-commands can be found at `Project > Source Documentation` [*see*].

– –

### *Sub-Command List*

```
E5MessageBox     Python re        QColorDialog     QFileDialog     QFontDialog
QInputDialog     QMessageBox      QRegExp          QRegularExpression
```

### *Viewpoint*

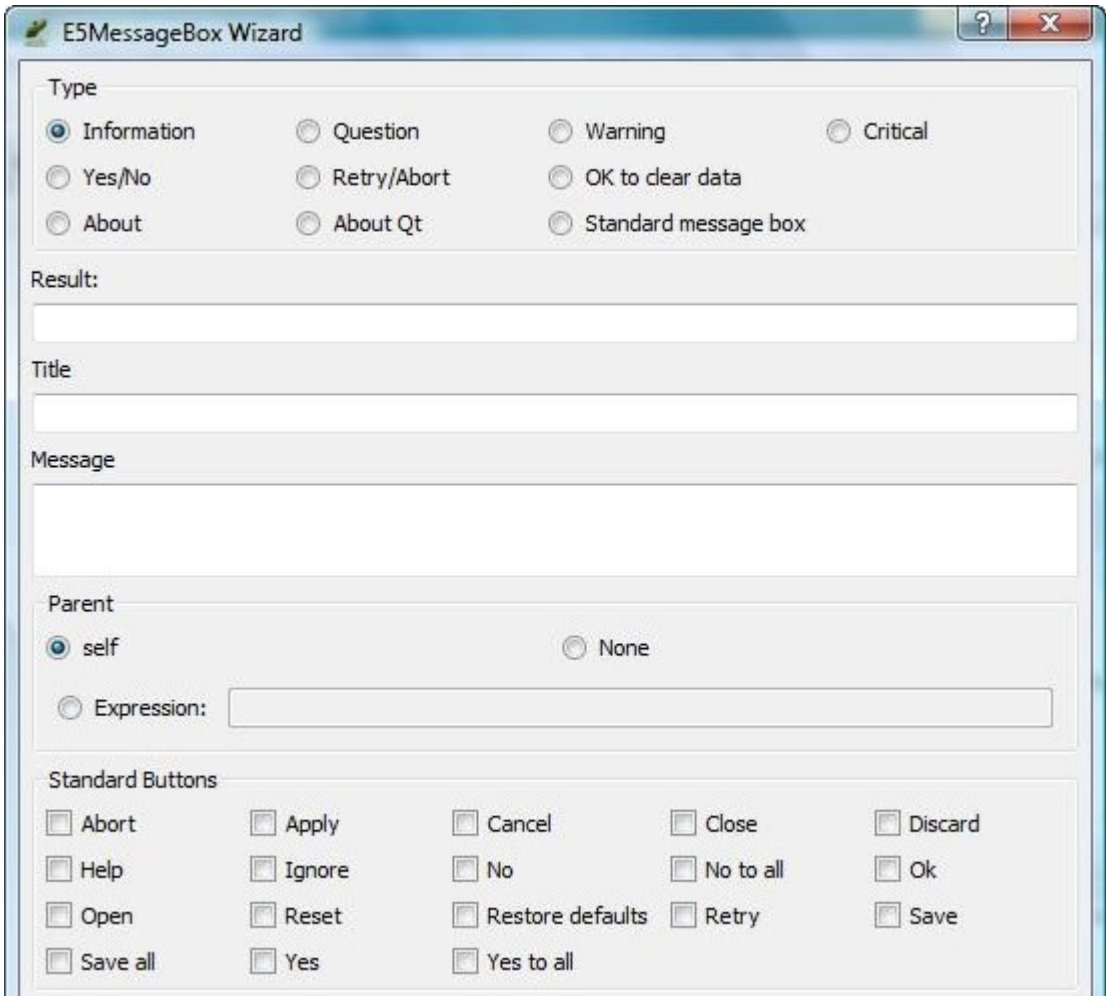
<~> In this set of sub-menu commands we'd suggest to drop the repetition of the term "Wizard", for manifest redundancy.

– –

74 Actually, all Core Plugin items [*cf.:* `Plugins > Plugin Infos...`].

Extras > Wizards > **E5MessageBox**
Extras > Wizards > **Python re**
Extras > Wizards > **QColorDialog**
Extras > Wizards > **QFileDialog**
Extras > Wizards > **QFontDialog**
Extras > Wizards > **QInputDialog**
Extras > Wizards > **QMessageBox**
Extras > Wizards > **QRegExp**
Extras > Wizards > **QRegularExpression**

A set of s/w productivity tools, offering such kind of "Wizard" dialog box:



Most of these software productivity tools are for PyQt[75] graphic library users, concerning `QtGui` module (with classes: `QColorDialog`, `QFileDialog`, `QFontDialog`, `QInputDialog`, `QMessageBox`) and `QtCore` module (with class: `QregExp`); so to conveniently test and then automatically generate such kind of source fragments:

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

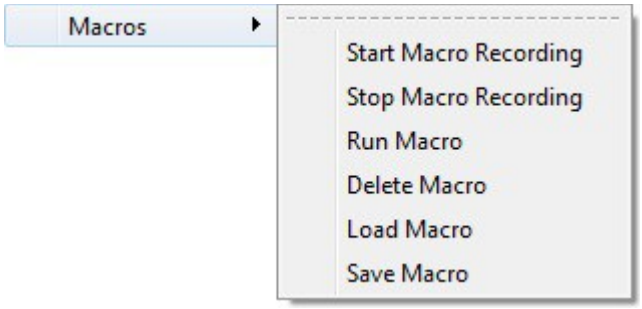
### *Viewpoint*

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

– –

## Extras > **Macros**

Designed to call a sub-menu of commands aimed at managing the Eric source text *Macros* [*cf.: L-Pane*: Template-Viewer, *in:* {4.West}*; a similar feature, more powerful*].




### *Remark*

This is a functionality derived from the QScintilla toolkit[76], offering the possibility of recording source text fragments on-the-fly, that is while writing them, so to possibly recall them later on, at will. Storage is user-based, so that each distinct user can organize his own personal Macros library.

#### *Sub-Command List*

```
Start Recording      Stop Recording      Run      Delete      Load      Save
```

### *Viewpoint*

<~> From these command names we've dropped the term “Macro”, for manifest redundancy.

75 About PyQt see: Prerequisites [*in:* {6.Trail}].

76 A Qt porting of Scintilla, as for www.scintilla.org [*see*].

We have successfully tested this functionality with such a frequently used source fragment:

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

that is a "Shebang" command, intended to be inserted as the first source line for Unix-like systems. This way saved as a source Macro, ready to be then recalled and inserted whenever needed.

– –

### Extras > Macros > **Start Recording**
### Extras > Macros > **Stop Recording**

Designed to start / stop the recording of a text editing session. A so recorded Macro can be then `Run` and `Saved` [*see hereafter*].

– –

### Extras > Macros > **Run**

To execute a just `Recorded` or a possibly `Loaded` Macro [*see*].

– –

### Extras > Macros > **Delete**

To delete from core memory a just `Recorded` or a possibly `Loaded` Macro [*see*]. Related Macro-file remains unaffected.

– –

### Extras > Macros > **Save**
### Extras > Macros > **Load**

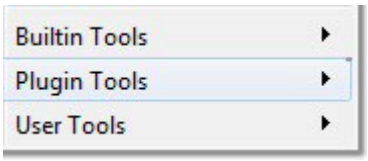
To save / load a possibly `Recorded` [*see*] Macro onto / from a disc file[77]. Then, so to be `Run`, a `Saved` macro must be `Loaded` first.

-<>-

77 Default directory: `<User>\_eric6`, file extension: "`*.macro`".

## Extras > **... Tools** Command Set

A set of commands for handling the distinct Eric's standard and custom *Tool Groups*; well worth to be first considered here together, as a coordinated set, before going specifically into each one of them [*see next sections*].




### *Specifications:*

`Builtin Tools` Standard Tool Group, pre-defined, populated and unalterable.
<。> A handy tool-shed of different tools [`Qt Designer`, ..., `Eric6 Web Browser`], conveniently located together and available from this single point [*and well worth knowing…*].

`Plugin Tools` Standard Tool Group, pre-created empty, custom populable via `Plugins > Install Plugins...` [*see*]. That is, a pre-arranged container for optional Eric Tools either ready available [*cf.: command* `Plugins > Plugin Repository...`] or possibly custom developed [*cf.: command* `Project > Packagers`]. Usable also via context command `Text Form (^) Tools` [*see*].

`User Tools` A collection of fully custom Tool Groups, initially empty. Can be user created, and then populated via `Extras > User Tools > Configure Current Tool Group...` [*see*].

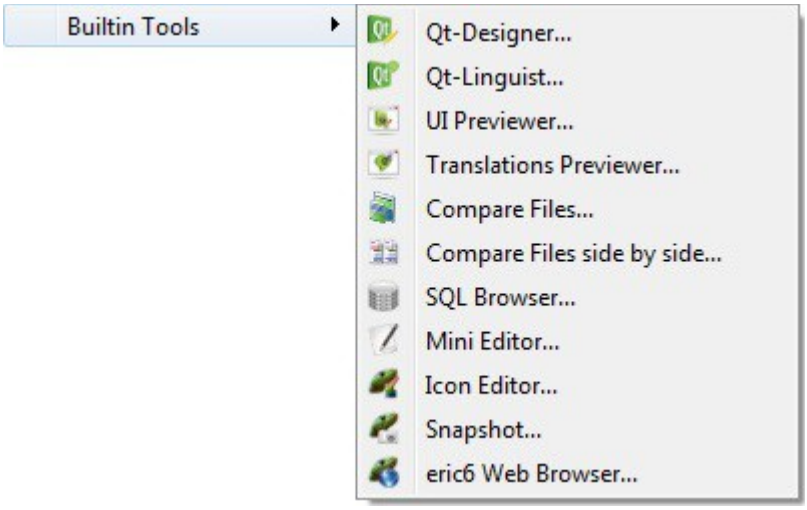
### *Remark*

<。> Builtin Tools is the only container of Eric Tools initially available and not-empty. The operative presence of the other two Plugin and User groups requires specific user interventions, as described hereafter [*see*].

-<>-

## Extras > **Builtin Tools**

Designed to call a sub-menu of commands aimed at executing specific utility programs, as hereafter listed.




### *Sub-Command List*

```
Qt-Designer...    Qt-Linguist...  UI Previewer...  Translations Previewer...
Compare Files...  Compare Files Side by Side...    SQL Browser...  Mini Editor...
Icon Editor...    Snapshot...     Eric6 Web Browser...
```

– –

## Extras > Builtin Tools > **Qt-Designer...**


A graphical user interface designer for Qt applications [*cf.:* `Forms (^) Open in Qt-Designer`].

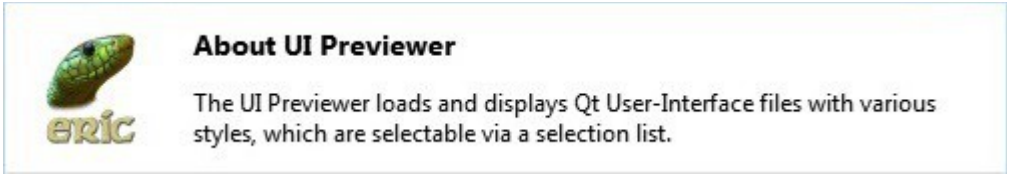
– –

## Extras > Builtin Tools > **Qt-Linguist...**


A tool for adding translations to Qt applications.

– –

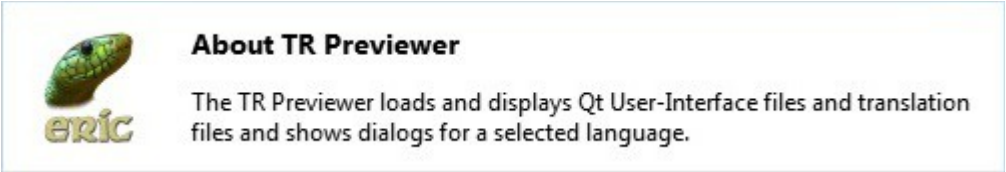
## Extras > Builtin Tools > **UI Previewer...**




A tool for loading and displaying Qt User-Interface files, with a selectable style.

– –

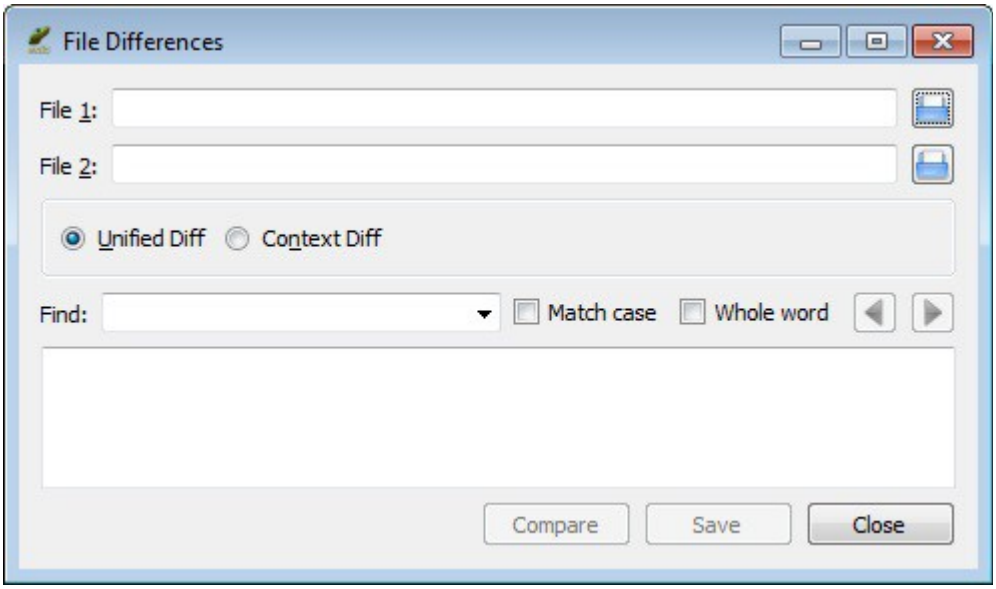
## Extras > Builtin Tools > **Translations Previewer...**




A tool for loading and displaying Qt User-Interface and translation files, with a selectable language.

– –

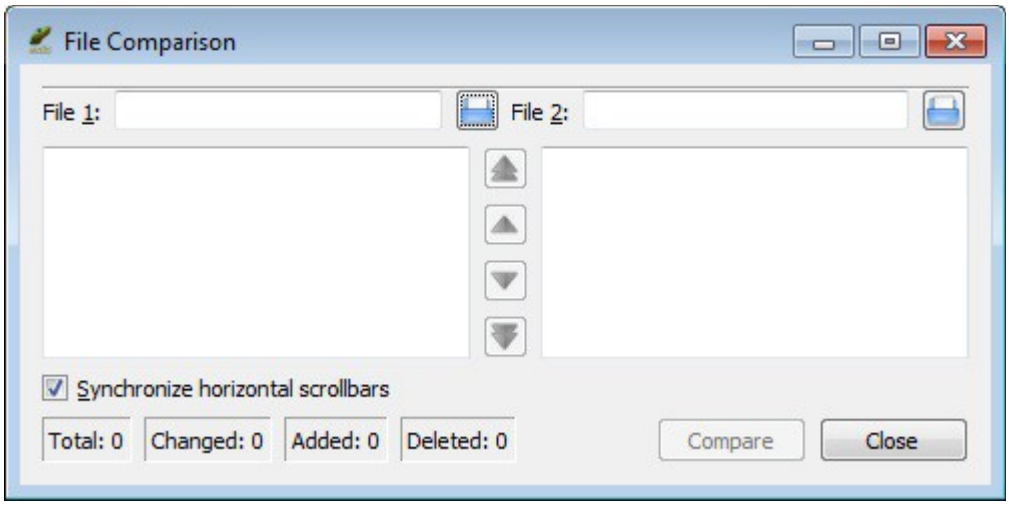
## Extras > Builtin Tools > **Compare Files...**




A tool for comparing text files, and showing differences.

– –

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




A tool for comparing text files, and showing differences on two distinct fields, side by side.

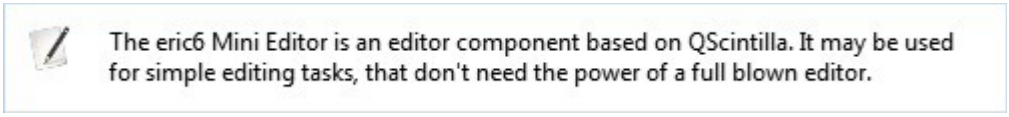
– –

## Extras > Builtin Tools > **SQL Browser...**


A tool to examine the data and the schema, and to execute queries on a SQL database.

– –

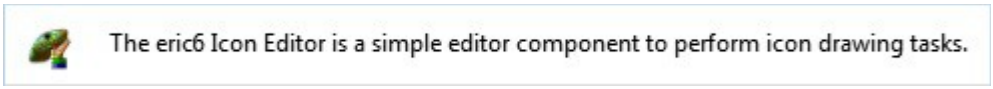
## Extras > Builtin Tools > **Mini Editor...**




A simple text editor, based on QScintilla.

– –

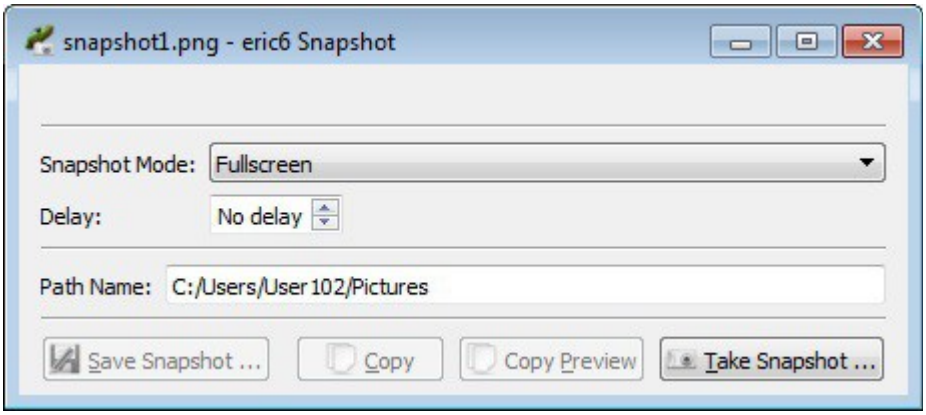
## Extras > Builtin Tools > **Icon Editor...**




A simple icon graphic editor.

– –

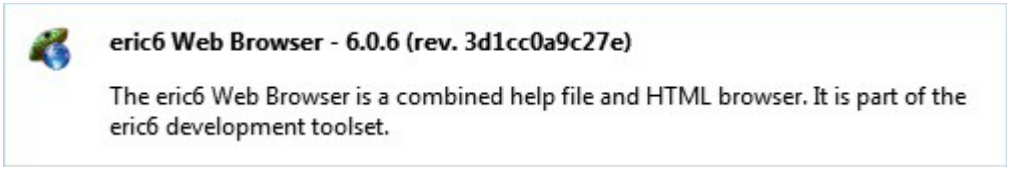
## Extras > Builtin Tools > **Snapshot...**




A tool for screen capture, that is for taking and saving screen image snapshots.

– –

## Extras > Builtin Tools > **Eric6 Web Browser...**




A combined help file and HTML browser. The very same tool that can be run with the equivalent `Help > Helpviewer...` commands [*see, and see also:* `Settings > Preferences... - Help > Help Viewers`]. Equivalent also in the sense that only one instance of this tool will execute at once, whatever the calling commands called.

– –

### *Remark*

In case you were surprised by such alphabetic icons scattered all over an Eric Web Browser page:


consider that these alphabetic icons are intended as an “Accessibility feature” meant to be shown / hidden acting on the “`Ctrl`” key as a toggle, and meant to show you which the “Access key” to press so to activate the related link. Anyhow a (`Ctrl+R`) “Reload” of the page, and you'll sweep them away.

–<>–

## Extras > **Plugin Tools**

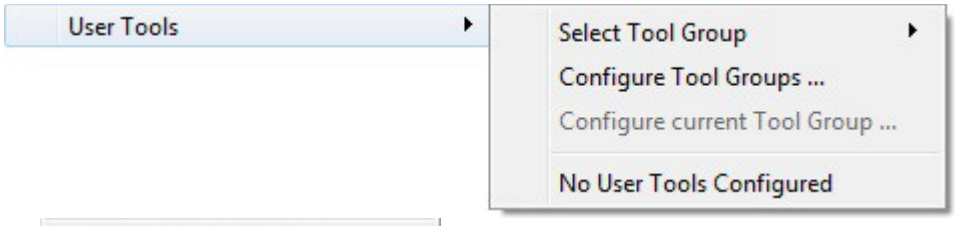
Designed to call a sub-menu of commands aimed at executing specific utility programs, field installable, such as the `Color String` here shown as an example.


For all available Plugin Tools, and how to get them, refer to the main command menu `Plugins` [*see, starting with:* `Plugins > Plugin Repository...`, *and then:* `Plugins > Install Plugins...`].

– –

## Extras > **User Tools**

Designed to call a sub-menu of commands aimed at managing and using Groups of custom User Tools.




The bottom item 

in this list plays the role of a status line, shown when no User Tool Group results defined [*cf.: next* `Configure Tool Groups...` *command*]. For a not-empty configuration see next `Extras > User Tools > Select Tool Group` command hereafter.

### *Sub-Command List*

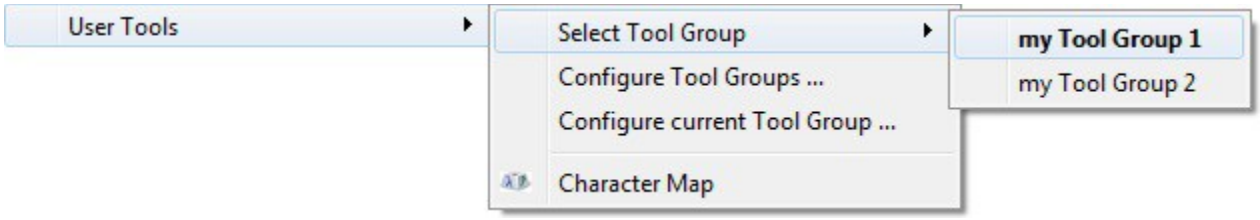
`Select Tool Group` `Configure Tool Groups...` `Configure Current Tool Group...`[78]

– –

78 <~> In the original syntax of these these last two commands there is an extra-blank inserted before the "..." ellipsis that, we think, should be eliminated. Ok, it's a minor formal imperfection, anyhow worth to be fixed.

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

Designed to select a given custom Tool Group, if any available, so to possibly manage it [*via:* `Configure Current Tool Group...`] and use its contents.




Selected group will be then appear bold-faced and its Tools, if any, become shown and usable as the "`Character Map`" in the above example.

– –

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

Designed to show a dialog box where to manage—create, delete, …—User Tool Groups. That is, further custom containers[79] for custom Tools, available at a per-User level, besides the two initial standard Builtin and Plugin[80] Tools containers, available at a per-Eric installation level.


79 Actually, not to be intended as file directories, but as sets of dedicated configuration data.
80 Standard Eric Groups that, by the way, will not here listed, where only the Custom Groups are.

*Specifications:*

`New` Reset entry field "`Group name`"

`Change` Rename selected Group to "`Group name`"

`…` All other controls here assumed as self-explanatory

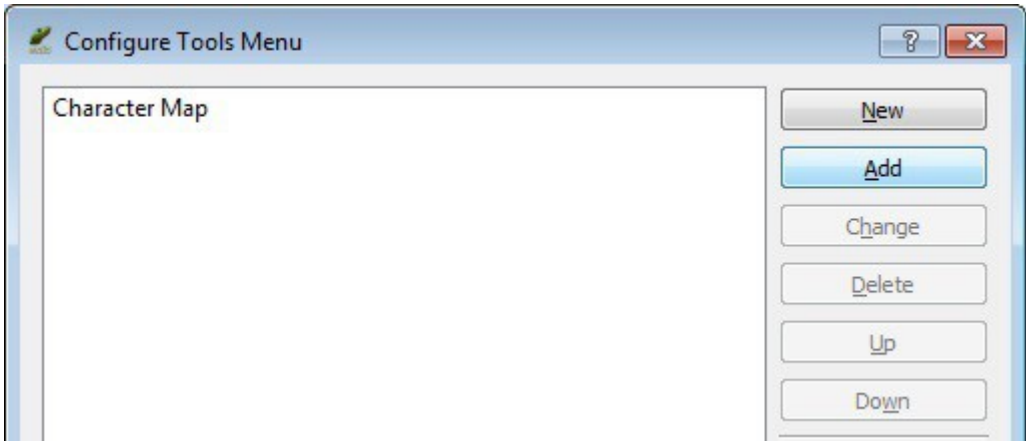
– –

Once such a custom "*`my Tool Group`*" has been added, it can be selected, configured [*cf. next:* `Extras > User Tools > Configure Current Tool Group...`]—that is, populated—and then used.

– –

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

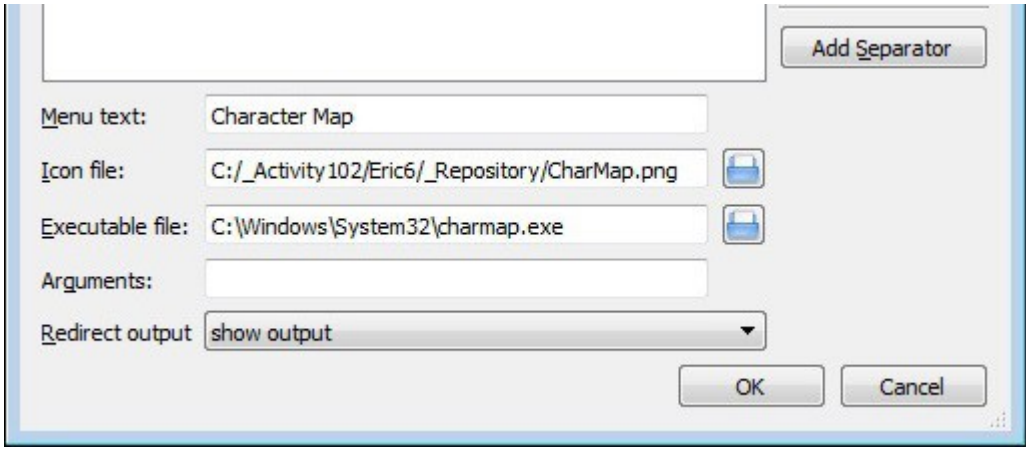
Designed to show a "`Configure Tools Menu`" control box where to populate[81] with the desired tools the currently selected custom User Tool Group; such as the "*`my Tool Group`*" example as previously created via `Extras > User Tools > Configure Tool Groups...` [*see*].




*Specifications:*

`New` Reset all entry fields

81 <~> We must confess that, at first, we remained rather confused by these two almost identically named commands: "`Configure Tool Groups...`" and "`Configure Current Tool Group...`"; until we realized that they simply deal with such plain concepts as "container" and "contents". <~> That's why here we explicitly used the term "populate", that is "fill", for clearness' sake; and certainly not "configure", which is such a generic term so to signify almost nothing.



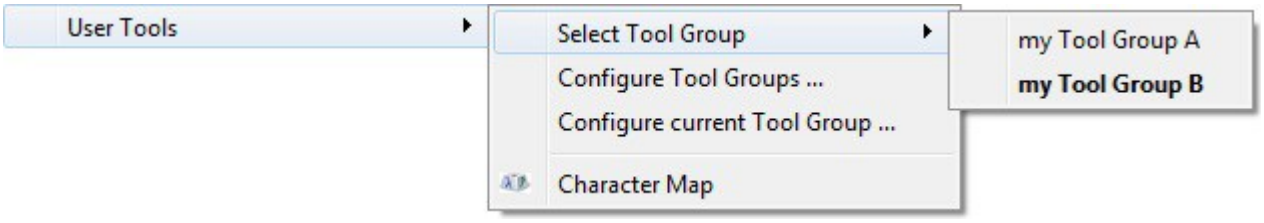

… All other controls here assumed as self-explanatory

## *Remark*

<!> Note that here, in the "`Executable file`" field, is where virtually any external program, that is any application available in the system in use can be entered, *becoming this way callable directly within Eric*.

– –

The "`charmap.exe`" application, here added as an example, will then become available to be called as "`Character Map`", at the bottom of the `User Tools > …` sub-menu set, provided the related custom User Tool Group, here: "`my Tool Group B`", is selected [*see*].



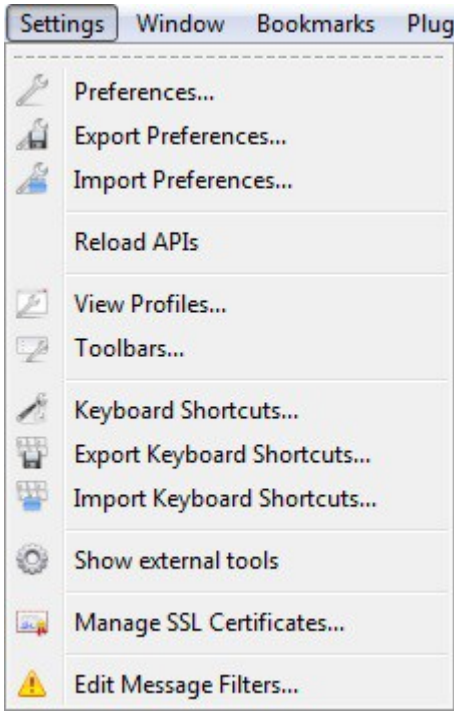

This `charmap.exe` is a system tool that has been here chosen as an example both for its simplicity and also because it may come actually handy for overcoming this tiny Eric flaw:

<~> The entering of a special character, that is a character that usually cannot be directly found on your keyboard, such as: `Alt 126`, for: "~", doesn't work well on the standard Eric text editor[82].

– –

82 <。> Anyhow, for an alternative Eric way of inserting "any" characters, see: *L-Pane*: Symbols [*in:* {4.West}].

# **Settings** Command Menu



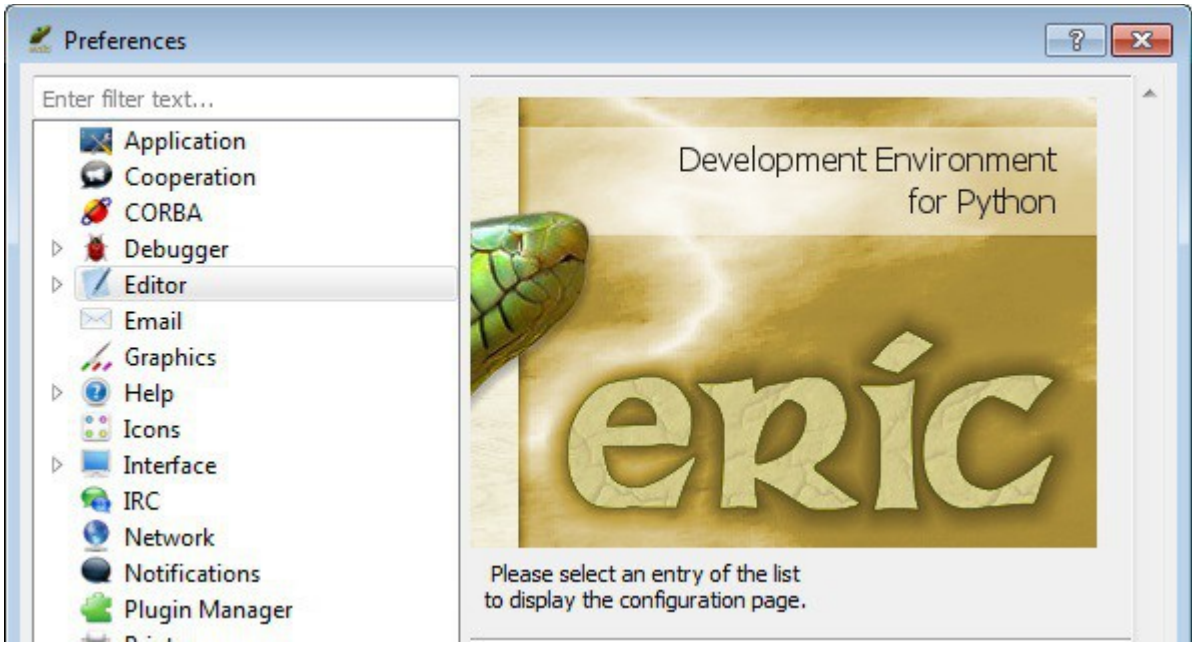

## *Command List*

```
Preferences...          Export Preferences...          Import Preferences...
Reload APIs             View Profiles...               Toolbars...
Keyboard Shortcuts...   Export Keyboard Shortcuts...   Import Keyboard Shortcuts...
Show External Tools     Manage SSL Certificates...     Edit Message Filters...
```

-<>-

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

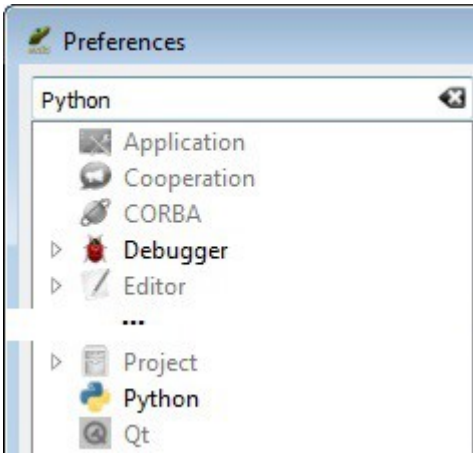
Designed to show a "`Preferences`" window aimed at managing the comprehensive set of Eric operative settings.




### *Specifications:*

`Enter filter text...`

Search engine within the Preferences tree. Where with, say, a "`Python`" pattern you'll get such items in evidence:



With the "`Debugger`" branch here set in evidence thanks to its contents, as can be verified expanding it[83]:


`Preferences tree`

Initially shown collapsed, then conveniently shown expanded as last set by the user.

– –

### *Remark*

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 here offer specific description where needed, on a case-by-case basis.

– –

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

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

Designed to export / import a custom Preference File containing all Eric preference settings.

### *Remark*

Such file appears to be organized into sections and keywords[84], not so difficult to recognize and interpret as a set of Eric preferences.

– –

83 [*Feature Under Revision*] Such expansion is scheduled to become automatic with next rev. 6.2.

84 Actually, it's a traditional Windows "`*.ini`"-like file, with default store location: `<Python3X>\...`

## Settings > **Reload APIs**

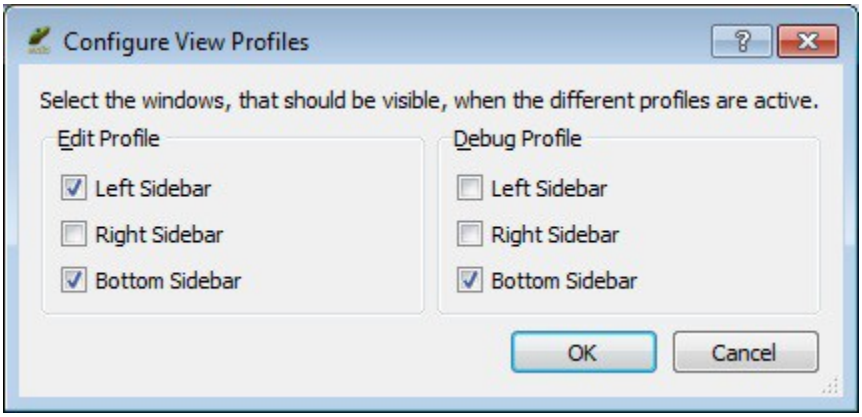
Designed to refresh the Coding Aids condition [*cf.:* `Edit > Complete`, Preliminary Explanations] as currently set on the "`Configure API files`" form of `Settings > Preferences...`; with compilation of the API info files included [*see also:* `Editor > APIs` *at* Eric Setup Completion, *in:* {6.Trail}].

Command useful to ensure the desired Coding Aids condition if accidentally altered, enabling it also for all Text Edit Forms possibly already opened.

– –

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

Command designed to show a control box where to define the two so called *View Profiles*[85] for the Edit and Debug operating modes, as selectable with the "`Window > Edit / Debug Profile`" commands [*see*].

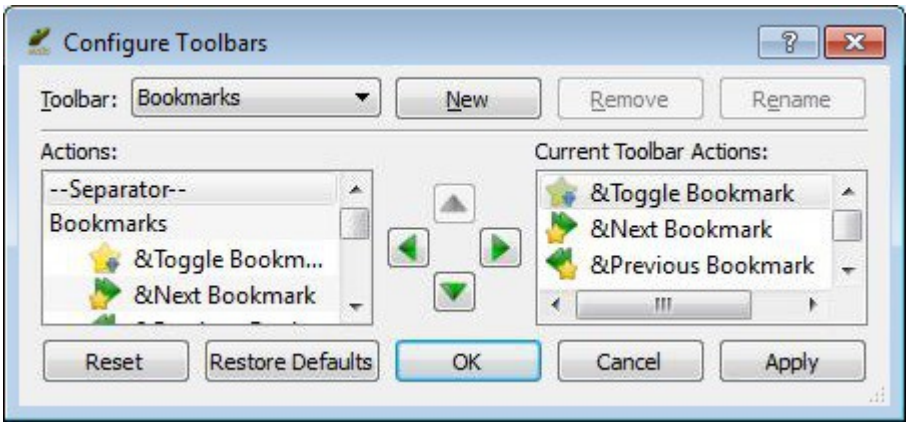



It's where to set which of the available *Auxiliary Forms* [*see:* Eric Window Map, *in:* {Map}] are to be displayed for the Edit and Debug operating modes. The overall Eric Window will then vary accordingly.

– –

85 *Profile*, that is a conventional term here adopted to signify: *Set of user selectable attributes*.

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



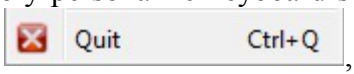

Designed to show a rich and friendly configurator box, available to inspect and personalize the Eric Tool Bar [*see:* Tool Bars *final section here in* {1.North}*, and* Eric Window Map *in* {Map}], so to possibly modify the default setting and adapt it to specific user's needs and preferences [*see also:* `Window > Toolbars`].

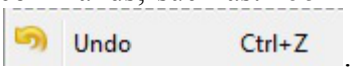
– –

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

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

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

Set of commands designed to activate a "`Keyboard Shortcuts`" configurator box, where to inspect and possibly personalize keyboard shortcuts associated to menu commands; such as: `Ctrl+Q` for `File > Quit` 

, or `Ctrl+Z` for `Edit > Undo` 

.

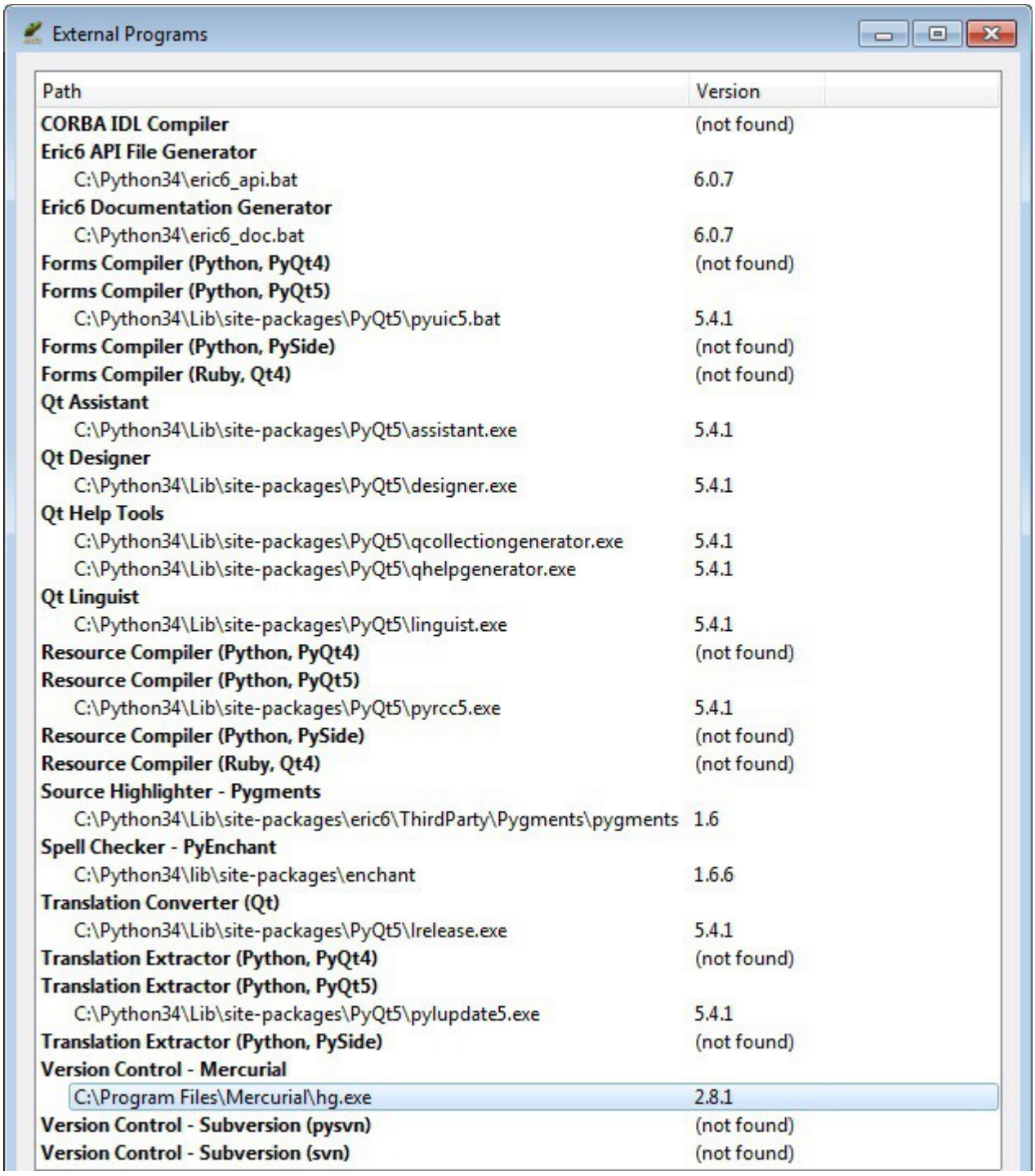
And then also to possibly save / restore all them into a dedicated configuration file.

### *Remark*

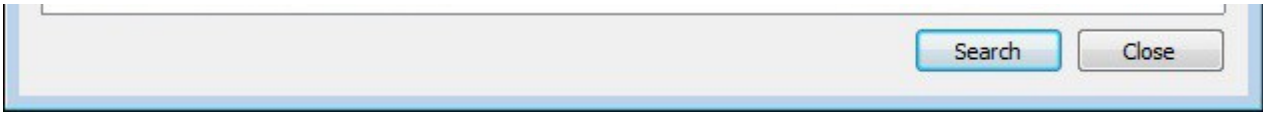
A configuration mechanism operating onto a specific XML file of extension "`*.e4k`" and with a reassuring standard "`default.e4k`" version located into the `<Python3X>\Lib\site-packages\eric6` directory [*see*]. That is, a file that could be wisely used as a safeguard mechanism, just in case you'd venture too far away from standard conditions.

–<>–

## Settings > **Show External Tools**



"`External Programs`" form, designed to show the whole set of "expected" optional programs and tools that Eric, as an IDE, is designed to "Integrate" into its single "Developing Environment". Each item on this list with specified its attributes of path and version, if actually installed, or a "`(not found)`" flag, if not. Precise list of all these items may vary when different plug-ins are present [*cf.:* Plugins Command Menu *section*].




### *Specifications:*

`Search` Button to refresh the list.

– –

## *Remark*

Note that in this list the Python and PyQt items aren't even shown, as mandatory prerequisites. Then the actual presence in this list of an optional item may be eider the result of a deliberate installation action or the automatic consequence of a prerequisite's package setup. For the distinction between mandatory and optional items see section Setup and General Management [*in:* {6.Trail}], and also command: `Help > Show Versions`.

## *Viewpoint*

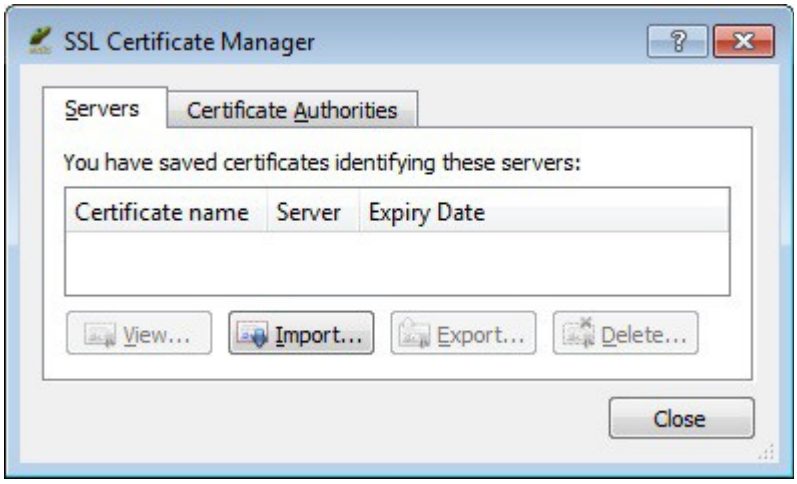
<~> A rather useful[86] list, worth to be watched over and analyzed so to better evaluate the Eric's potential available to the user.

-<>-

86 <!> and impressive

## Settings > **Manage SSL Certificates...**

As with any other application possibly connected in the Internet, Eric too might have to perform "secure" transactions, via HTTPS protocol and with Secure Sockets Layer (SSL) Certificates. Here is where such SSL Certificates can be managed.




It's the same tool you'll find also with the Eric's web browser [*see:* `Help > Helpviewer...`], with both tools operating on the same Certificate store, at first initialized via PyQt[87].

– –

### *Viewpoint*

No further detailed description of this tool will be here offered, as:

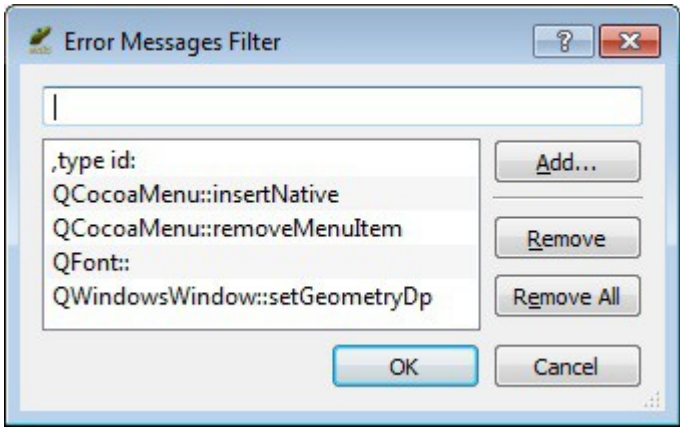
- This is a subject already very well covered on all major web browser [*see any...*].
- This is anyhow a largely automatized technology, rarely requiring a direct user interventions; at least at the client's side.

– –

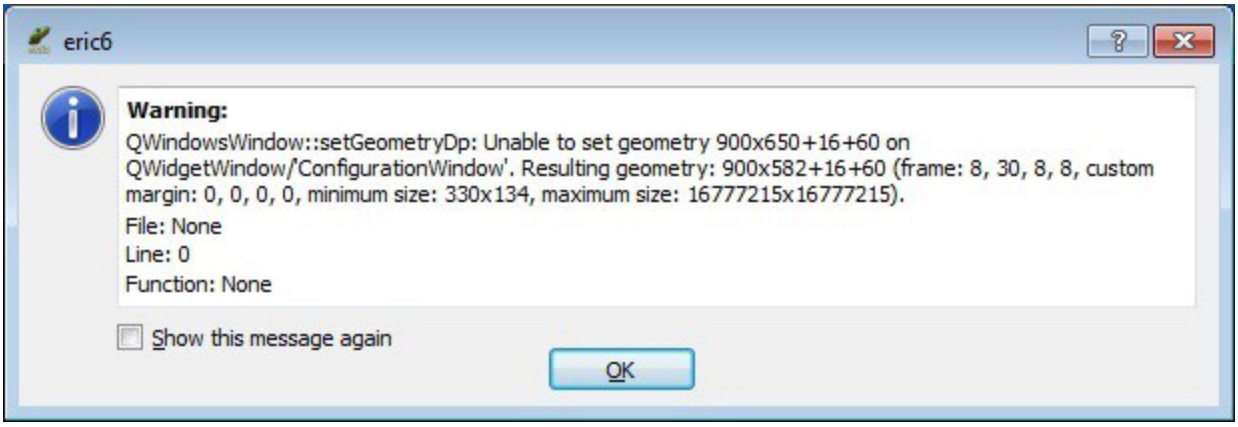
## Settings > **Edit Message Filters...**

Designed to filter out some messages possibly generated by the PyQt library engine, and that the Eric user doesn't want to see any more.

87 About the PyQt Secure Sockets Layer (SSL) protocol handling you may refer to the QtNetwork module classes and OpenSSL Toolkit.



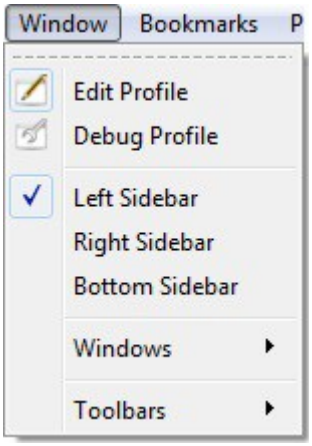

This feature is provided as a remedy at Eric's level to a little PyQt's inconvenience as, in certain circumstance, it keeps showing such warning messages as the one hereafter [*see*], even though explicitly requested by the user, on its own dialog boxes, not to show it any more.




Just adding a "`QWindowsWindow::setGeometryDp`" entry into the "`Error Messages Filter`" list would get you rid of future show of this warning message.

-<>-

## **Window** Command Menu




### *Command List*

```
Edit Profile       Debug Profile
Left Sidebar       Right Sidebar      Bottom Sidebar
Windows            Toolbars
```

– –

### Window > **Edit Profile**

### Window > **Debug Profile**

*Profile* Conventional term here adopted to signify: Set of attributes, user selectable.

Commands designed to switch between the two distinct *Edit* or *Debug* View Profiles for the current Eric work session, corresponding to the two named different activities; that is: Edit / Debug.

Possibly different Auxiliary Forms [*cf.:* `Settings > View Profiles...`] and Tool Bars [*cf.:* `Window > Toolbars`] will consequently be shown.

#### *Viewpoint*

<~> Here it would be really nice to be visually informed about which is the Profile currently enabled, for instance disabling its related command, as currently dummy, and keeping enabled only the other one, as currently effective.

That would be useful, for instance, to be well informed which Profile will be affected by possible changes executed on the current Toolbar settings, as indeed there are two of them, one for each Profile. By the way, a little useful choice not so evident, and worth to be known [*cf.:* `Window > Toolbars`].

– –

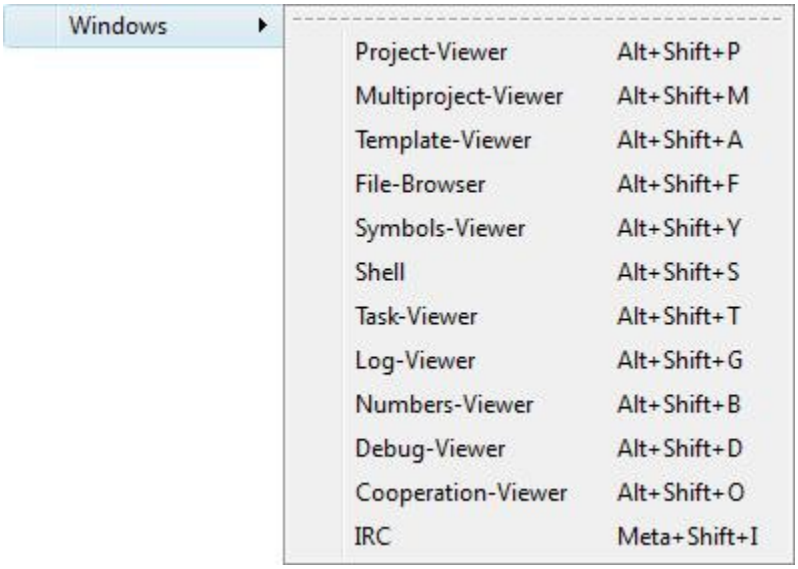
## Window > **Left Sidebar**
## Window > **Right Sidebar**
## Window > **Bottom Sidebar**

Designed to show / hide the named Eric's Auxiliary Forms on the respective Left / Right / Bottom Panes [*see section:* Eric Window Map, *in:* {Map}]. The overall Eric Window will vary accordingly.

– –

## Window > **Windows**

Designed to activate a sub-menu where to chose which one of the listed control forms to activate and show, on the respective hosting window pane.

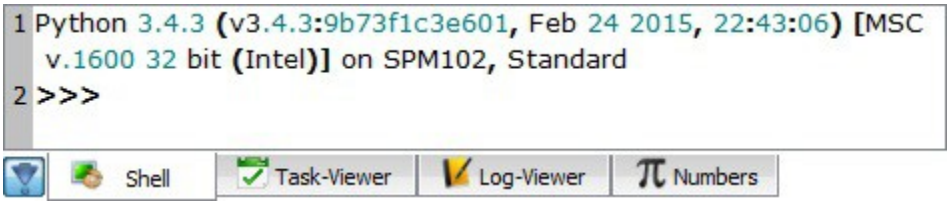



Same effect as a "`(^)`"[88] click on the related tab, when in sight. That is, tab: [`Project-Viewer`, …, `Symbols`] at the left, [`Shell`, …, `Numbers`] at the bottom, [`Debug-Viewer`, …, `IRC`] at the right side.

– –

The related left / bottom / right hosting window pane and Auxiliary Form will be shown, if not already in display [*cf. commands:* `Window > Left / Right / Bottom Sidebar`]; e.g., with `Windows > Shell`:

88 "`(^)`" is the symbol here adopted for a "click" action [*see:* Typographical Conventions, *in:* Appendix].

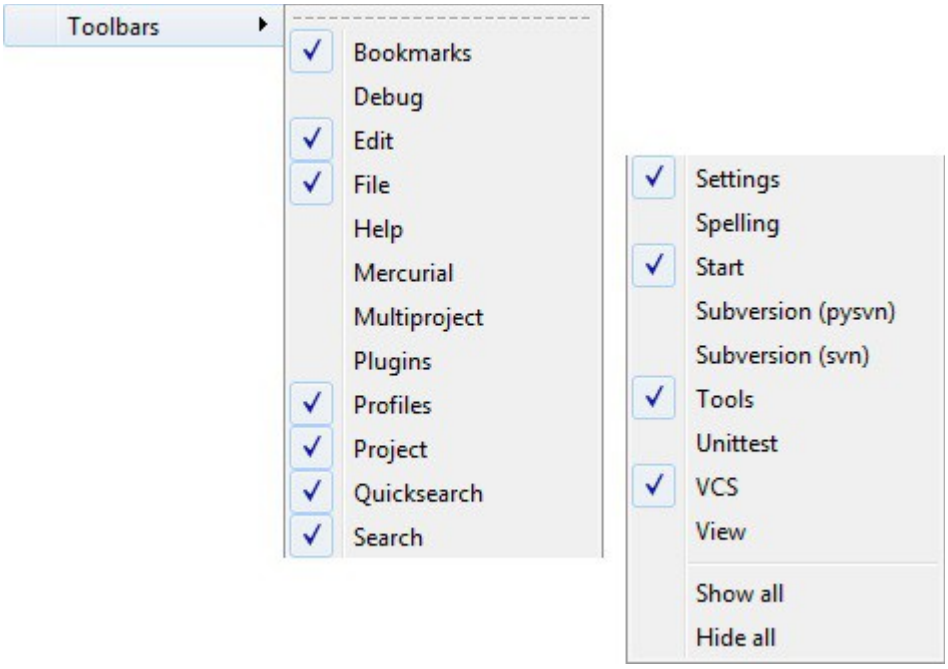



Detailed description of each one control form is hereafter offered on sections: Eric – South / West / East Side [*in:* {3.South}, {4.West}, {5.East}].

– –

## Window > **Toolbars**

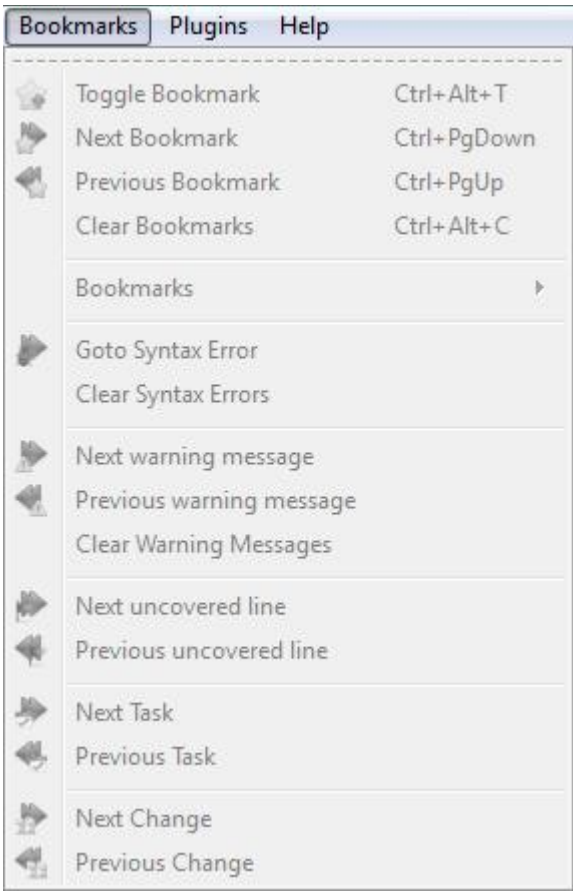
Designed to show a sub-menu where to set the show / hide status for each and all Eric "Tools"—such as: `Bookmarks`, `Debug`, …, `View`—available on the Tool Bars for the current Profile of use; that is either Edit or Debug [*cf.:* `Window > Edit / Debug Profile`].




Tools can be custom configured with `Settings > Toolbars` [*see*]. Here in display the initial default condition [*cf. also:* Tool Bars, *final section here in:* {1.North}].

–<>–

# **Bookmarks** Command Menu



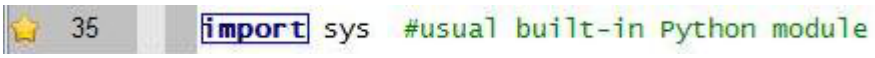

## *Command List*

```
Toggle Bookmark      Next Bookmark            Previous Bookmark   Clear Bookmarks
Bookmarks            Goto Syntax Error        Clear Syntax Errors
Next Warning Message Previous Warning Message Clear Warning Messages
Next Uncovered Line  Previous Uncovered Line
Next Task            Previous Task            Next Change         Previous Change
```

– –

## Bookmarks > **Toggle Bookmark**

Designed to set / unset a bookmark at the current pending line, on the active source text. 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.

### *Remark*

Command `Project > Session` [*see*] is provided to save / recall all bookmarks currently defined for a given Project.

– –

## Bookmarks > **Next Bookmark**

## Bookmarks > **Previous Bookmark**

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

– –

## Bookmarks > **Clear Bookmarks**

Designed to clear all bookmarks currently defined.

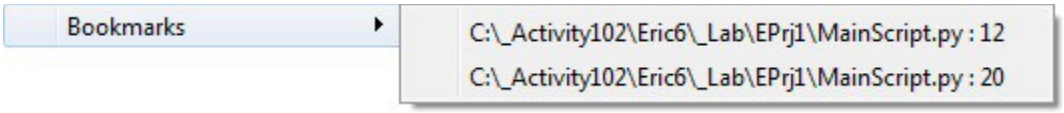
### *Remark*

Useful here to note that the command `Project > Session` [*see*] is provided to save / recall all bookmarks currently defined for a given Project.

– –

## Bookmarks > **Bookmarks**

Designed to list all and, possibly, jump to one of the currently available source text bookmarks [*cf.:* `Bookmarks > Next / Previous Bookmark`].

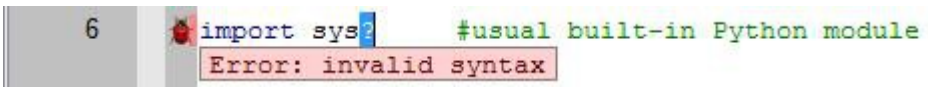



– –

## Bookmarks > **Goto Syntax Error**

Designed to jump to the possible syntax error diagnosed on the current source module (only). This command, when found "dim"-disabled, implies that no syntax error has been so far detected on the current module.

Syntax errors take precedence over all possible code warnings, and are detected by the Python syntax checker which stops operating at any such first error detected. The offending line is "bug"-marked in the Notice column, that is the rightmost on the vertical Ruler Bar [*cf.:* Text Edit Central Pane, *in:* {2.Central}].




This marking takes place at loading and saving of the related module, and also at typing of source text, after an idle time, configurable[89] [*see:* `Settings > Preferences... – Editor > Syntax Checker, Configure Syntax Checker settings`].

### *Remark*

<。> This Notice column marking can be cleared with the `Bookmarks > Clear Syntax Errors` command [*see*], and will be anyhow conveniently updated and refreshed automatically while typing.

– –

89 Note that this idle time starts to be measured when the operator pauses, so not to disturb a normal editing session.

## Bookmarks > Clear Syntax Errors

Designed to clear all possible syntax errors in display on different modules [*see also:* `Bookmarks > Goto Syntax Error`]. Next `File > Save` command will anyhow refresh the Python syntax checking, and all the consequent error display.

– –

## Bookmarks > Next Warning Message

## Bookmarks > Previous Warning Message

Designed to jump to the next / previous warning message currently shown [*cf. also:* `Bookmarks > Clear Warning Messages`].


Warning conditions are checked by Python only if no syntax error is detected. All suspicious lines are ⚠-"Warning" marked in the Notice column, the rightmost on the vertical Ruler Bar [*cf. also:* `Bookmarks > Goto Syntax Error`].

### *Remark*

Actual display of such messages can be inhibited un-ticking the "`Show annotations`" check-box in the "`Configure editor style`" form [*see:* `Settings > Preferences... – Editor > Style`]. Anyhow note that the presence of such related Notice icon ⚠, as well the action of these two commands, will remain unaffected.

– –

## Bookmarks > Clear Warning Messages

Designed to clear all possible warning messages currently shown [*cf. also:* `Bookmarks > Next / Previous Warning Message`].

### *Remark*

Messages that anyhow are conveniently refreshed at any subsequent text editing action and on-the-fly `File > Save` action [*cf.:* `Bookmarks > Goto Syntax Error`].

– –

## Bookmarks > Next Uncovered Line

## Bookmarks > Previous Uncovered Line

Designed to move the text editor's insertion cursor to the next / previous source code line currently marked as "uncovered", as the result of a "Coverage Run" [*see: commands* `Start > Coverage Run` *and* `(^) Show > Show Code Coverage Annotations`].

– –

## Bookmarks > Next Task

## Bookmarks > Previous Task

Designed to jump to the next / previous line of the current source module[90] marked with such usual "*Extracted Task*"[91] comment markers: "`#TODO:`" and "`#FIXME:`".


These lines are detected when source modules are saved or opened, and are also marked with a "✓" tick-sign icon in the Notice column; that is the rightmost column on the vertical Ruler Bar [*see*].

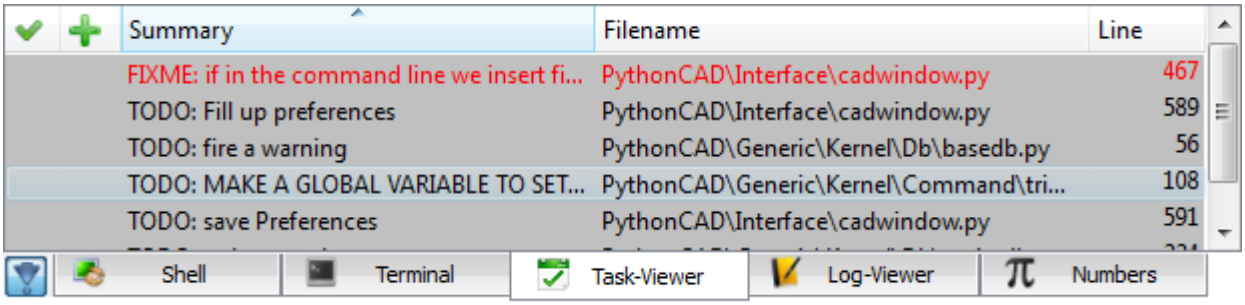
<!> Note that only the Extracted Task-markers typed ***exactly*** as in the related `Settings > Preferences... – Tasks, Tasks Markers` will be detected [*verify*].

– –

90 <~> Current module only, alas. It would be nice if this command's scope were extended to all modules in a Project.

91 "Extracted Task", not to be confused with the different "Manual Task" which can be created with the context command `Task (^) New Task...` [*see: section* Bt-Pane: Task-Viewer*, in:* {3.South}].

These same Tasks can also be inspected, and manually created and managed, in the "`Task-Viewer`" of the tabbed Bottom Form [*see in:* {3.South}]; where, in case of a Project, they are all conveniently listed, independently from the opening state of the respective module belonging to the project.




## *Remark*

With Eric Projects, such Tasks are also recorded into a dedicated "`*.e4t`" file, located into the standard Project management sub-directory `<Project>\_eric6project`.[92]

– –

## Bookmarks > **Next Change**

## Bookmarks > **Previous Change**

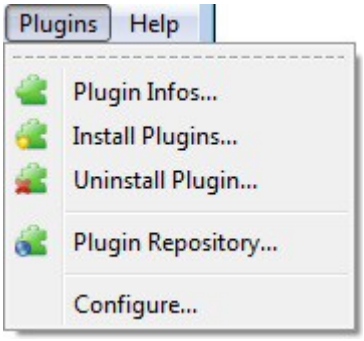
Designed to jump to the next / previous line of the current source text changed during the current edit session, and consequently side-marked as in this example.

```
7   import sys                  #usual built-in Python module
8   from PyQt4.QtGui import *
9   app = QApplication(sys.argv)
```

–<>–

92 Whereas the other so called Global Tasks are recorded in a `<User>\_eric6\eric6tasks.e4t` file [*for details see:* ***Bt-Pane:*** Task-Viewer, *in:* {3.South}].

# **Plugins** Command Menu



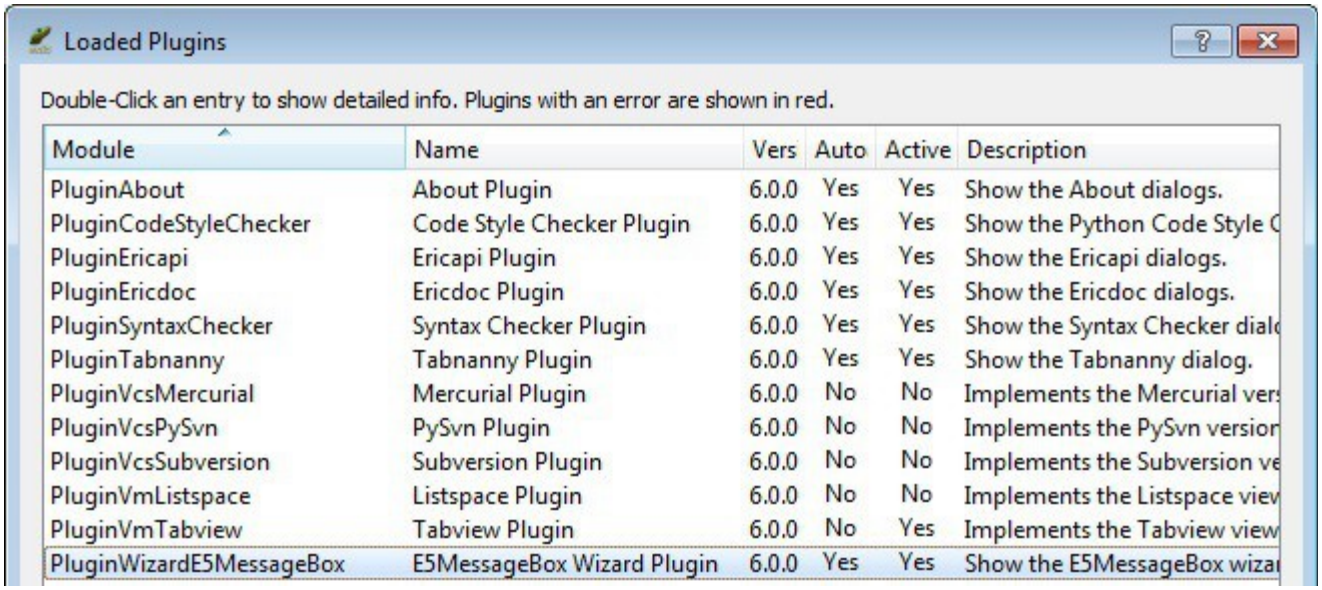

### *Command List*

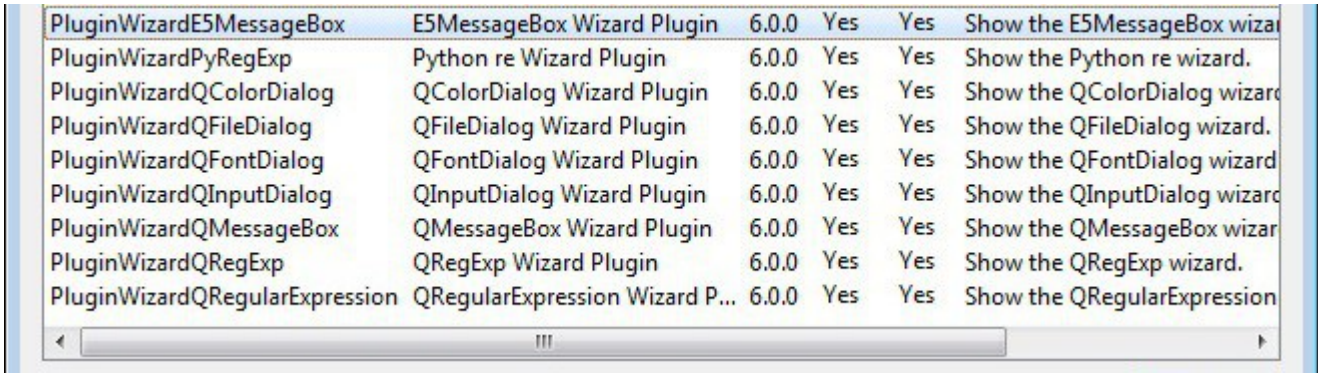
`Plugin Infos...` `Install Plugins...` `Uninstall Plugin...` `Plugin Repository...` `Configure...`

– –

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

Designed to show a comprehensive "`Loaded Plugins`" table form, where are listed and briefly commented all Plugins currently available locally for Eric [*cf.: next command* `Plugins > Plugin Repository...` *for all Plugins available on-line for down loading, and* `Project > Packagers` *for custom created Plugins*].

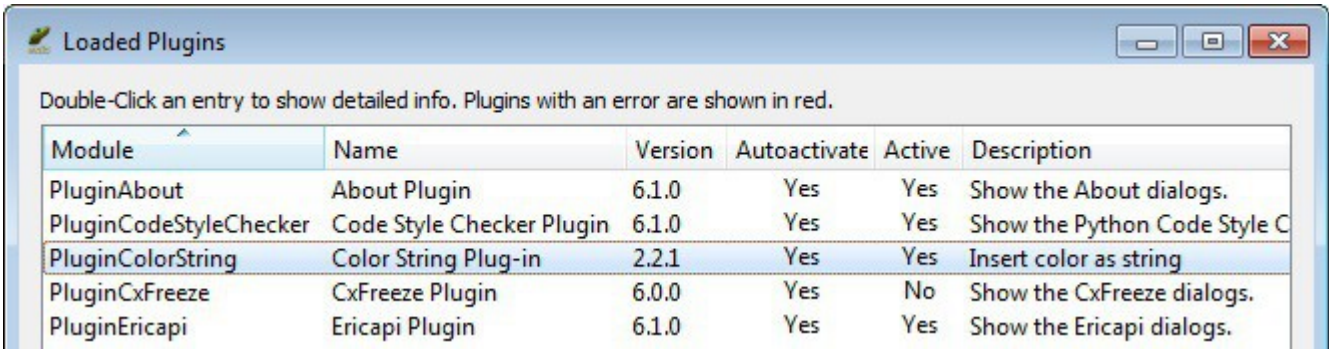




| | | | | | |
|---|---|---|---|---|---|
| PluginWizardE5MessageBox | E5MessageBox Wizard Plugin | 6.0.0 | Yes | Yes | Show the E5MessageBox wizar |
| PluginWizardPyRegExp | Python re Wizard Plugin | 6.0.0 | Yes | Yes | Show the Python re wizard. |
| PluginWizardQColorDialog | QColorDialog Wizard Plugin | 6.0.0 | Yes | Yes | Show the QColorDialog wizard |
| PluginWizardQFileDialog | QFileDialog Wizard Plugin | 6.0.0 | Yes | Yes | Show the QFileDialog wizard. |
| PluginWizardQFontDialog | QFontDialog Wizard Plugin | 6.0.0 | Yes | Yes | Show the QFontDialog wizard |
| PluginWizardQInputDialog | QInputDialog Wizard Plugin | 6.0.0 | Yes | Yes | Show the QInputDialog wizard |
| PluginWizardQMessageBox | QMessageBox Wizard Plugin | 6.0.0 | Yes | Yes | Show the QMessageBox wizar |
| PluginWizardQRegExp | QRegExp Wizard Plugin | 6.0.0 | Yes | Yes | Show the QRegExp wizard. |
| PluginWizardQRegularExpression | QRegularExpression Wizard P... | 6.0.0 | Yes | Yes | Show the QRegularExpression |

This "`Loaded Plugins`" list comprises:

- Initial set of the so called "*Core Plugins*", that is all add-ons made available by a standard Eric setup, as described in some detail hereafter [*cf.: next* ***Core Plugin*** *list*], along with the specific Eric command making use of each of them; e.g.: `PluginAbout`, by `Help > About eric6`, or `PluginCodeStyleChecker`, by `Project > Check > Code Style...`
- "*Plugin Tools*" out of the Plugin Repository [*cf.: next* `Plugins > Plugin Repository...` *command, and section* Plugin Tools and Commands] which, when installed [*cf.: next:* `Plugins > Install Plugins...`], result listed as in the following "`PluginColorString`" example, intermixed in pure alphabetic order amongst the other Plugins[93].



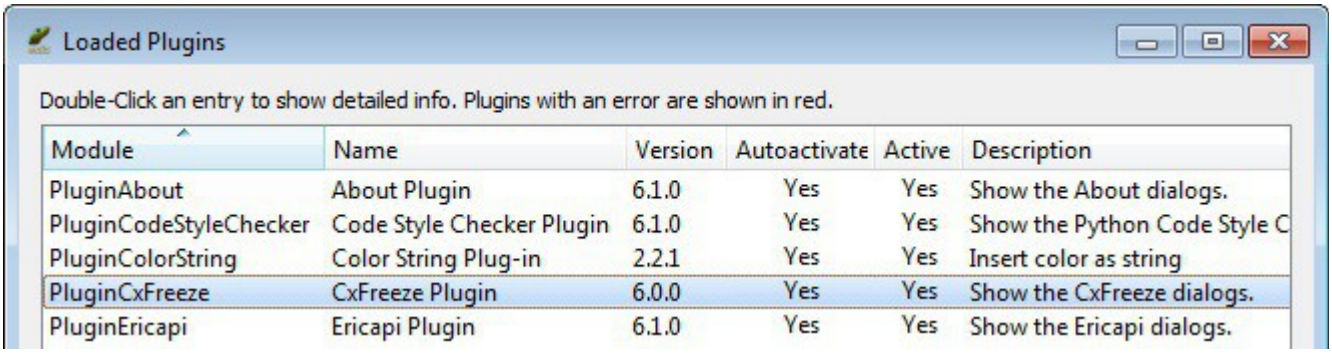

| Module | Name | Version | Autoactivate | Active | Description |
|---|---|---|---|---|---|
| PluginAbout | About Plugin | 6.1.0 | Yes | Yes | Show the About dialogs. |
| PluginCodeStyleChecker | Code Style Checker Plugin | 6.1.0 | Yes | Yes | Show the Python Code Style C |
| PluginColorString | Color String Plug-in | 2.2.1 | Yes | Yes | Insert color as string |
| PluginCxFreeze | CxFreeze Plugin | 6.0.0 | Yes | No | Show the CxFreeze dialogs. |
| PluginEricapi | Ericapi Plugin | 6.1.0 | Yes | Yes | Show the Ericapi dialogs. |

And then available as new Eric commands, located into the `Extras > Plugin Tools` sub-menu group [*see, there with this* `Color String` *example – see also: context* `Text Form (^) Tools`].

- By the way, note that the other two Tool Groups treated by the `Extras > … Tools` command, that is: Builtin Tools—such as: `Qt-Designer`, …, `Eric6 Web Browser`—and the User Tools [*see*], are here not considered.

93 <~> A bit confusing, isn't it?

- "*Plugin Commands*" out of the same Plugin Repository [*cf. above point*] which, when installed, result listed as in the following "`PluginCxFreeze`" example, intermixed in pure alphabetic order amongst the other Plugins[94].




  And then available as new Eric commands variously located somewhere[95] on the standard menu; such as the example here offered at: `Project > Packagers > Use cxfreeze` [*see*].

## *Remark*

More precisely, this is a tool fully available and operable only in presence of the "real" CxFreeze package installed, being this `PluginCxFreeze` simply an interface for running it within Eric [*see installation example at:* `Plugins > Install Plugins...`, *along with related* Remark].

– –

### *Core Plugins*

<。> Core Plugins constitute a permanent extension of Eric's base functionality, being used as engines by some standard commands; as hereafter listed [*see*].

– –

`PluginAbout` by `Help > About eric6`

`PluginCodeStyleChecker`
by `Project > Check > Code Style...`, and the corresponding `Project-Viewer: Sources (^)` and `Text Form (^) Check >` ... context commands

`PluginEricapi`
`PluginEricdoc` by `Project > Source Documentation >` ...

94 <~> A bit confusing, isn't it?

95 <~> Different target locations for different commands, conveniently preset on a case-by-case basis, here not detailed and left to be "discovered" by the adventurous user [*sorry…*].

`PluginSyntaxChecker`
: by `Project > Check > Syntax...`, and the corresponding `Project-Viewer: Sources (^)` and `Text Form (^) Check > ...` context commands

`PluginTabnanny`
: by `Project > Check > Indentations...`, and the corresponding `Project-Viewer: Sources (^)` and `Text Form (^) Check > ...` context commands

`PluginVcsMercurial`
`PluginVcsPySvn`
`PluginVcsSubversion`
: as Eric's VCS interface engines [*see:* `Project > Version Control`, *in* {1.North}]

`PluginVmListspace`
`PluginVmTabview`
: by `Settings > Preference... – Interface > Viewmanager`, for `Listspace` / `Tabbed View`

`PluginWizardE5MessageBox`
`...`
`PluginWizardQRegularExpression`
: for all `Extras > Wizards` sub-commands.-

– –

## Plugins > **Install Plugins...**

## Plugins > **Uninstall Plugin...**

Designed to show a control form where to possibly install / uninstall Plugin Tools and Commands downloaded from the Eric Web Repository [*see:* *command* `Plugins > Plugin Repository...`]. Installed Plugin Tools will add to the `Extras > Plugin Tools` sub-menu [*see*], whereas Plugin Commands will appear as new commands variously added to the standard Eric menu.

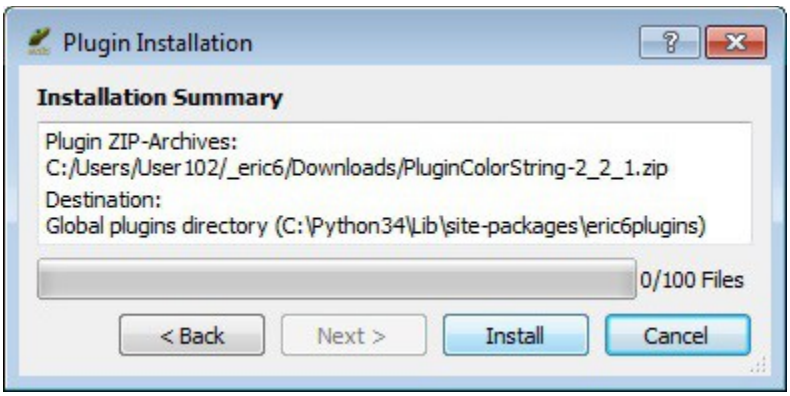
– –

An install example both of a Plugin Tool, "`Color String`", and a Plugin Command, "`CxFreeze`", will be hereafter offered [*see*].

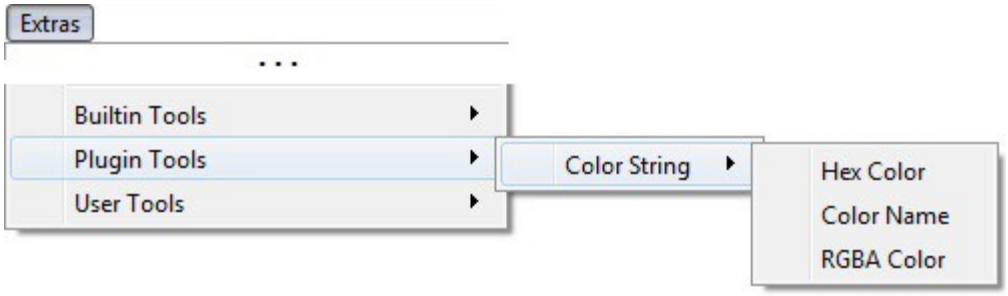
-<>-

## *Install Example of Plugin Tool:* `Color String`

Once downloaded as a `PluginColorString-2_2_1.zip` file from the `Plugin Repository` [*see*], chosen the usability scope "`Global`", instead of (current-)"`User`", the related installation process will smoothly reach the condition hereafter summarized.



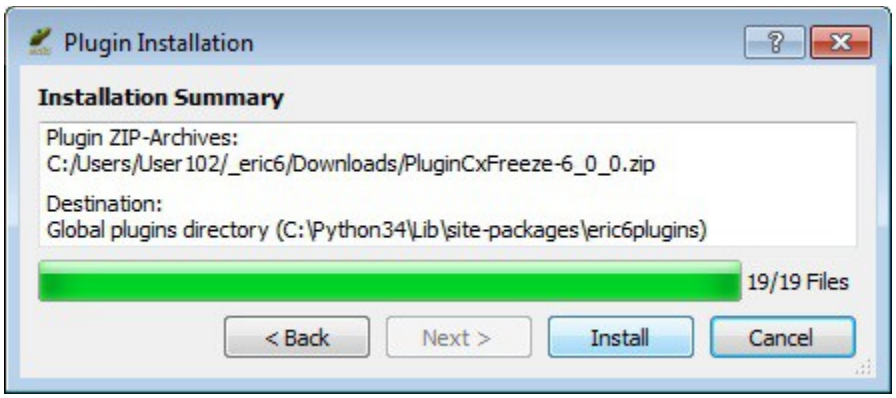

In conclusion, leading then to the following addition both into the `Extras > Plugin Tools` sub-menu and the context `Text Form (^) Tools` [*see*].




-<>-

## *Install Example of Plugin Command:* `CxFreeze`

Once downloaded as a `PluginCxFreeze-6_0_0.zip` file from the `Plugin Repository` [*see*], chosen the usability scope "`Global`", instead of (current-)"`User`", the related installation process will smoothly reach the condition hereafter summarized.




After such an `Install` action this Plugin will become usable with the new "`Use cx_Freeze`" command added at the bottom of the `Project > Packagers` set of sub-commands[96] [*see*],


and could also be spotted as a new item added on the `Plugins > Plugin Infos...` table; where, if right-clicked, it will show a pop-up menu to: `Show Details`, `Activate`, `Deactivate` [*see*].

<!> Note that, as in this `cx_Freeze` example, the installation of a Plugin Command can imply considerable modifications on the Eric's user interface; such as: additional menus, new menu entries and even new window panes.

– –

96 Well enabled and available also for any Project Type, not only for the "Eric Plugin" ones, as with the other `Packagers > Create` ... sub-commands [*see*].

## *Viewpoint*

<~> As a rule in this Report we consider *exclusively the original user interface* consequent a standard Eric setup, as shown in the Eric Window Map [*see in:* {Map}]; possibly leaving the treatment of special Eric features and configurations, such as those consequent the installation of a Plugin Command, to other dedicated documents [*see:* Scope of this Report, *in:* {0.Lead}].

– –

## *Prerequisite*

Anyhow note that this `CxFreeze` plugin is NOT an operative tool in itself, in fact it simply offers an interface front-end integrated into Eric for calling the original cx_Freeze tool[97], here assumed as already installed and available into the current system, and known to the user.

Indeed, these the required prerequisite actions:

- `http://cx-freeze.sourceforge.net/index.html` is where the required cx_Freeze item can be downloaded from; as here the: `cx_Freeze-4.3.4.win32-py3.4.exe`
- Usual set-up
- Then you'd better give a look at the directory: `<Python3X>\Scripts`
  so to verify that a `cxfreeze.bat` has been there created ok, at the cx_Freeze setup;
- If not[98], you should locate into the same directory the "`cxfreeze-postinstall`" Python script and run it by hand, so to get the required `cxfreeze.bat` generated.-

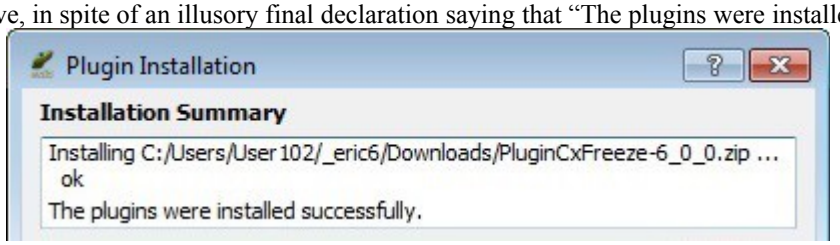
<.> This Plugin Command example has been here deliberately chosen as a comparatively complex one, so to illustrate various aspects of the subject.

-<>-

97 cx_Freeze is a known cross platform set of scripts and modules for "freezing" Python scripts into executable programs, distributed under the PSF open-source license.

About this tool as currently available in Eric, a preliminary "Eric (5) Project Packager – Technical Report" has been produced, now available still *AsIs* [*see:* Foreword, *What & Where in the Internet*].

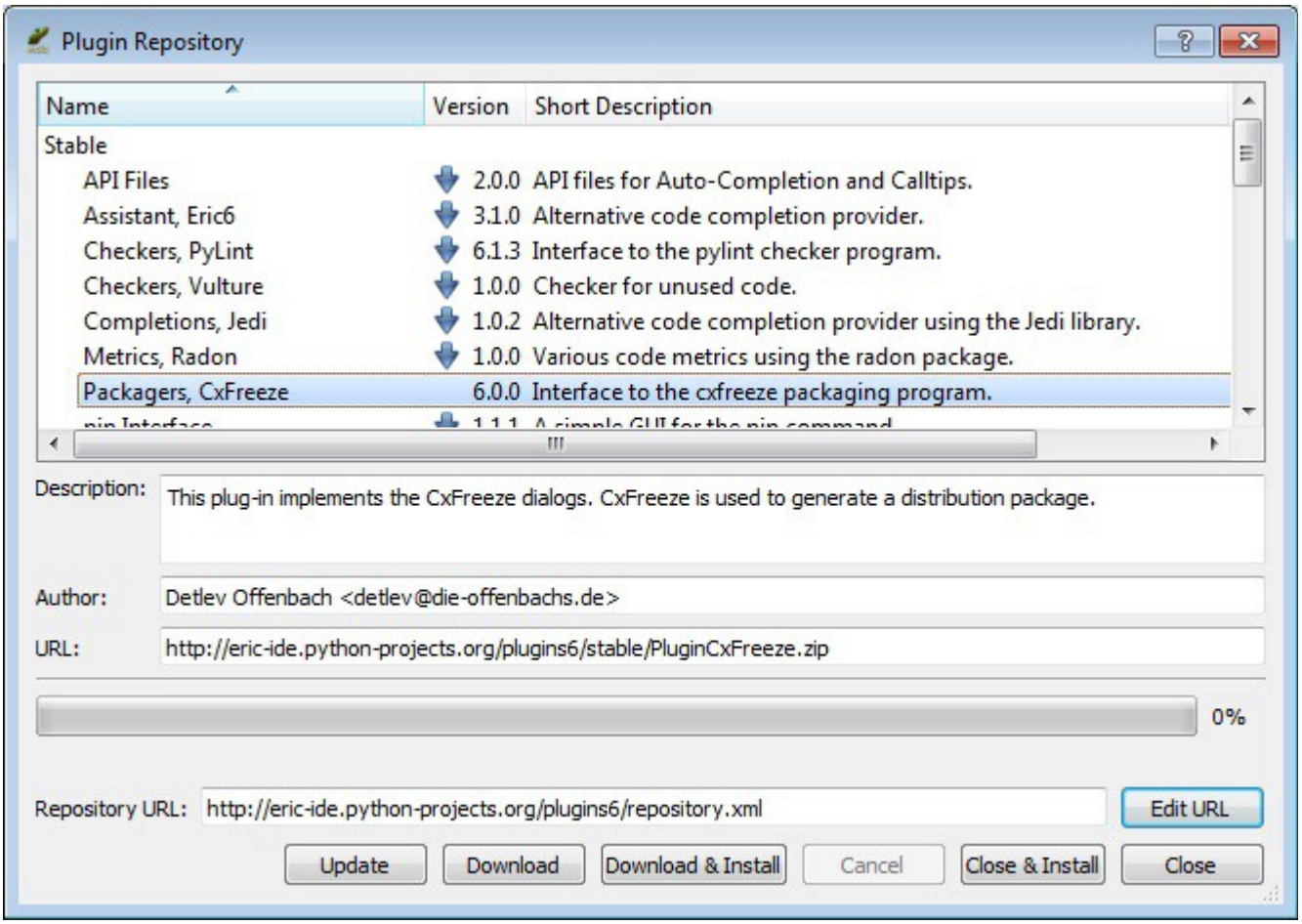
98 As we happened to have, in spite of an illusory final declaration saying that "The plugins were installed successfully":




<~> By the way, it's our opinion that an installation process should always give a proactive look at its prerequisites, so to fairly inform the possibly unaware Mr. User if any problem.

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

Designed to show a dialog box where to inspect, manage and, possibly, update the local list of Eric Plugins currently available for downloading on the dedicated Eric's Plugins Web Repository[99] [*then see also: command* `Plugins > Plugin Infos...` *for Plugins already available locally, and* `Project > Packagers` *for custom created Plugins*].



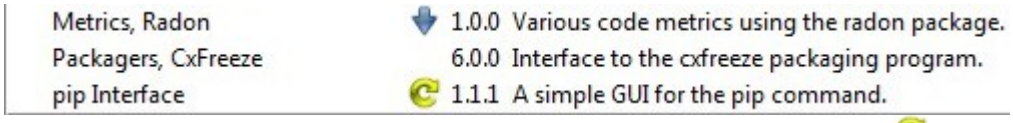

All controls of this box are here assumed as clear enough not to require any further explanations. `Update` and `Download` actions require an Internet connection active.

– –

99 Current URL: http://eric-ide.python-projects.org/pluginsX/repository.xml as here preset on the "`Repository URL`" field of this box.

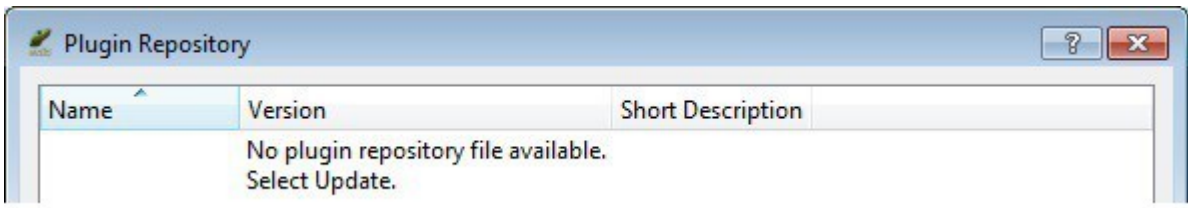
*Specifications:*

- The top-left "`Stable`" label [*see*] is to qualify this Repository, as opposed to another one provided for items "Under Development". Then the "`Description`" field is shown for the selected item.
- Down-loaded items go to the current local Repository: `<User>\_eric6\Downloads` [*cf.:* `Plugins > Configure...`].

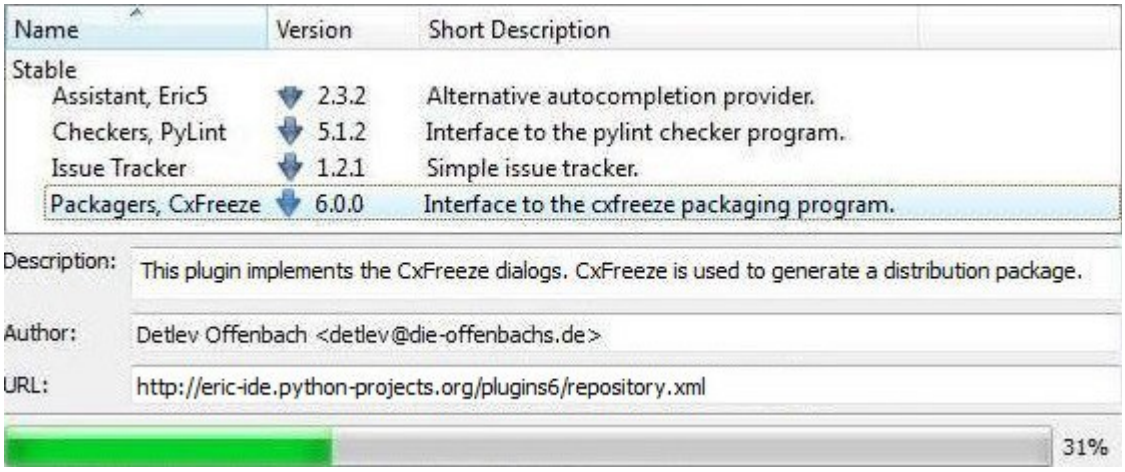



downward-arrow symbol is for items "`Download`"–available, changed to a curly arrow when downloaded; then missing when installed. Re-established when a new version for the same item is available for downloading.

- This the initial off-line aspect of the `Plugin Repository` form, when no related info is yet locally available[100].




This the same dialog box as it appears after being "`Updated`", during a "`Download`" action[101].




– –

100<。> Actually, in itself it is such a XML text file: "`<User>\_eric6\PluginRepository`"

101Example centered onto the `CxFreeze` package, for its relevance [*cf.:* `Plugins > Install Plugins...`].

### *Plugin Tools and Commands*

All items in this Repository, once downloaded, are meant to be installed so to become usable into Eric [*see:* `Plugins > Install / Uninstall Plugin...`]; some as Plugin Tools [*see:* `Extras > Plugin Tools` *sub-menu*], all the others as Plugin Commands [*see: preceding* `Plugins > Plugin Infos...` *command*].

<。> At present, these are the Plugin Repository items meant to become usable as Plugin Tools:

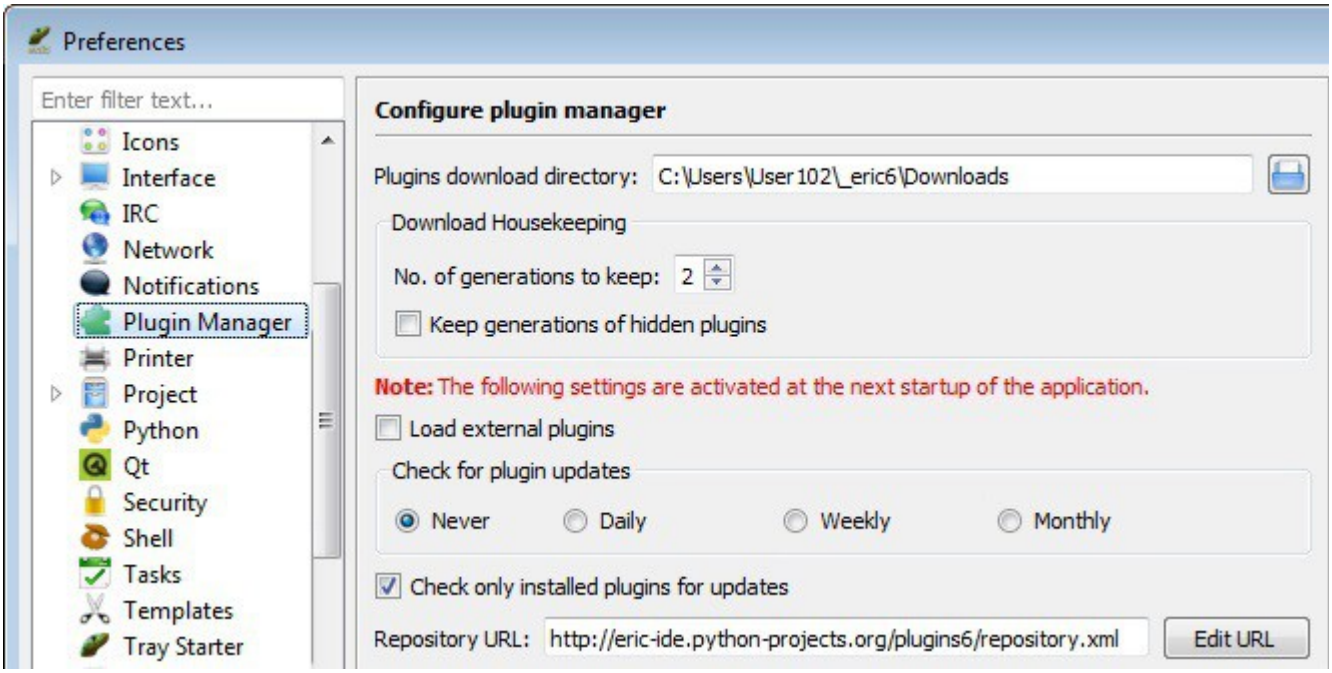
| | | |
|---|---|---|
| `Project, Web` | as: | `Web` |
| `Tool, Camel Case Handling` | as: | `CamelCase Handling` |
| `Tool, Color String` | as: | `Color String` |
| `Tool, Hash Generation` | as: | `Generate File Hash` |
| | and: | `Generate Directory Hash` |
| `Tool, Invert Hex Color` | as: | `Invert Hex Color...` |
| `Tool, Print Remover` | as: | `Remove Outputs` |
| `Tool, PySide to PyQt Converter` | as: | `PySide to/from PyQt` |
| `Tool, Selection Encloser` | as: | `Enclose Selection` |

A short description of each item is available both on this Plugin Repository and then also on the Loaded Plugins form [*see*].

-<>-

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

Designed to show the same control form as with command `Settings > Preferences...` [*see*], but opened directly onto the "`Plugin Manager`" section.



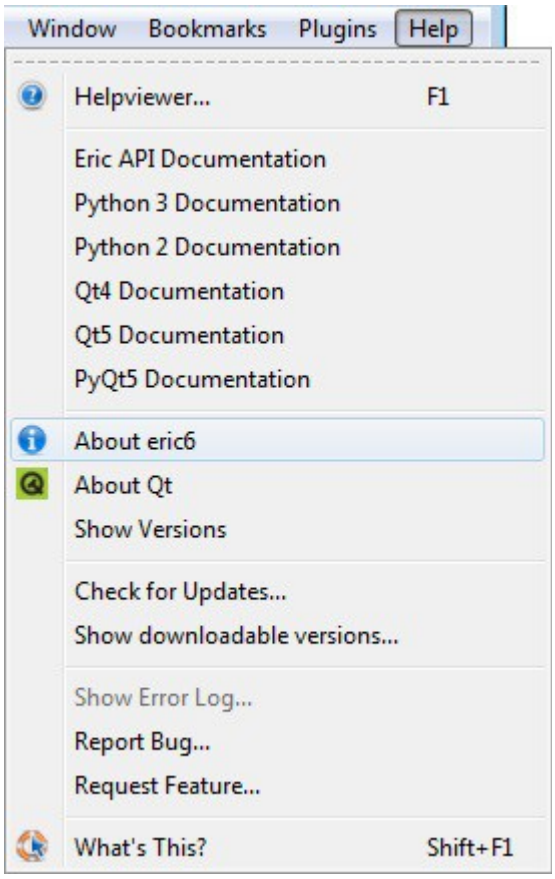

Note, in particular:

`Plugins download directory`
: To set the destination directory for down-loaded Eric Plugins, as for menu command `Plugins > Plugin Repository... > Update` [*see*].

`Load external plugins`
: To specify plugins from the possibly custom `Repository URL`, instead of those originally and automatically bound up with Eric.

-<>-

# **Help** Command Menu



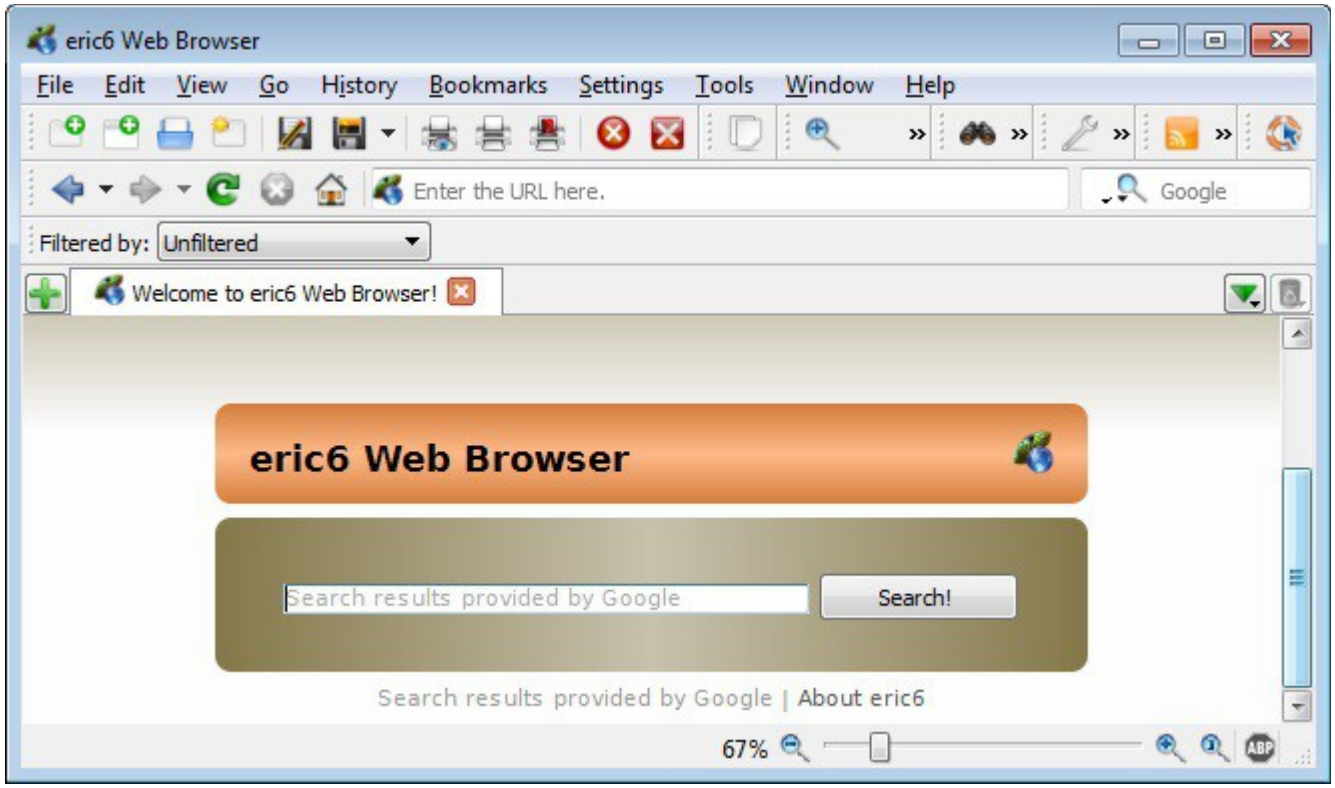

## *Command List*

```
Helpviewer...                Eric API Documentation     Python 3 Documentation
Python 2 Documentation       Qt4 Documentation          Qt5 Documentation
PyQt5 Documentation          About Eric6                About Qt
Show Versions                Check for Updates...       Show Downloadable Versions...
Show Error Log...            Report Bug...              Request Feature...
What's This?
```

– –

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

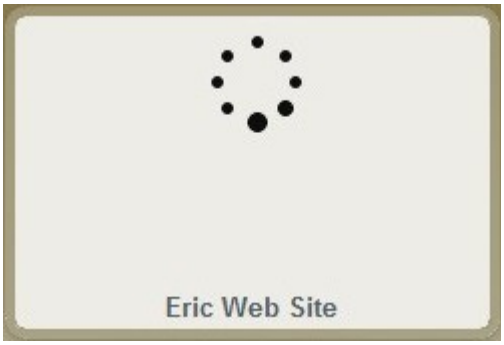
Designed to run the dedicated "Eric6 Web Browser" program which, besides being a web browser, is primarily aimed at browsing all Eric help material [*cf. also:* `Extras > Builtin Tools > Eric6 Web Browser...`].



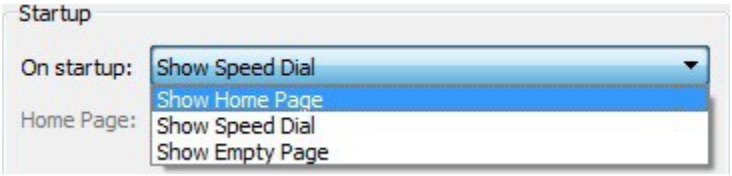

It is basically a full-fledged Web Browser, even if not comparable with the major browsers currently in use, which may be called either explicitly, with this command, or automatically, when inspecting the Help material here available; as, for instance, with the `Help > Eric API Documentation` command [*see hereafter*].

– –

<~> When called explicitly, the initial home page will typically show the "`Speed Dial`" thumbnails to pre-selected web sites [*see:* `Settings > Preferences... – Help > Eric6 Web Browser`, *the area:* `On startup`]. Therefore if, as it may well be, there is no active Internet connection, this could lead to such a standstill condition:




To avoid such an inconvenience you may instead simply preset the cited "`On start up`" preference to: `Show Home Page` [*cf. also:* Eric Setup Completion, *in* {6.Trail}].



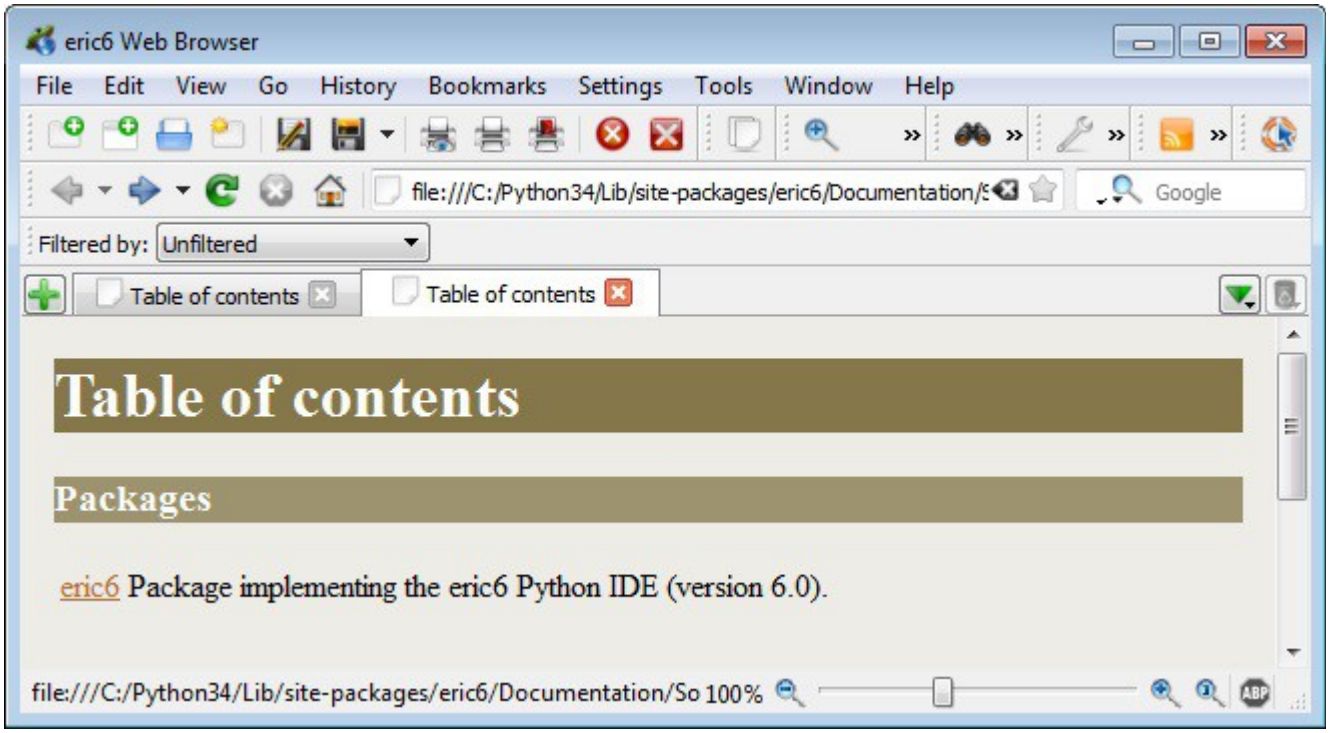

### *Viewpoint*

This is a self-standing subject of particular relevance, here just hinted at, as assumed off scope for the current Report [*see:* Scope of this Report, *in:* {0.Lead}]. Detailed description of this application can be found under the title "Eric 4 Web Browser — Technical Report" in its virtually identical Eric 4 version, on the Eric web site [*see:* Foreword, *What & Where in the Internet*].

– –

## Help > **Eric API Documentation**

Designed to show on the Eric Web Browser window the standard Python "documentation strings" of the API modules associated with the very development of the Eric application [*cf.: command* `Project > Source Documentation`].




## *Viewpoint*

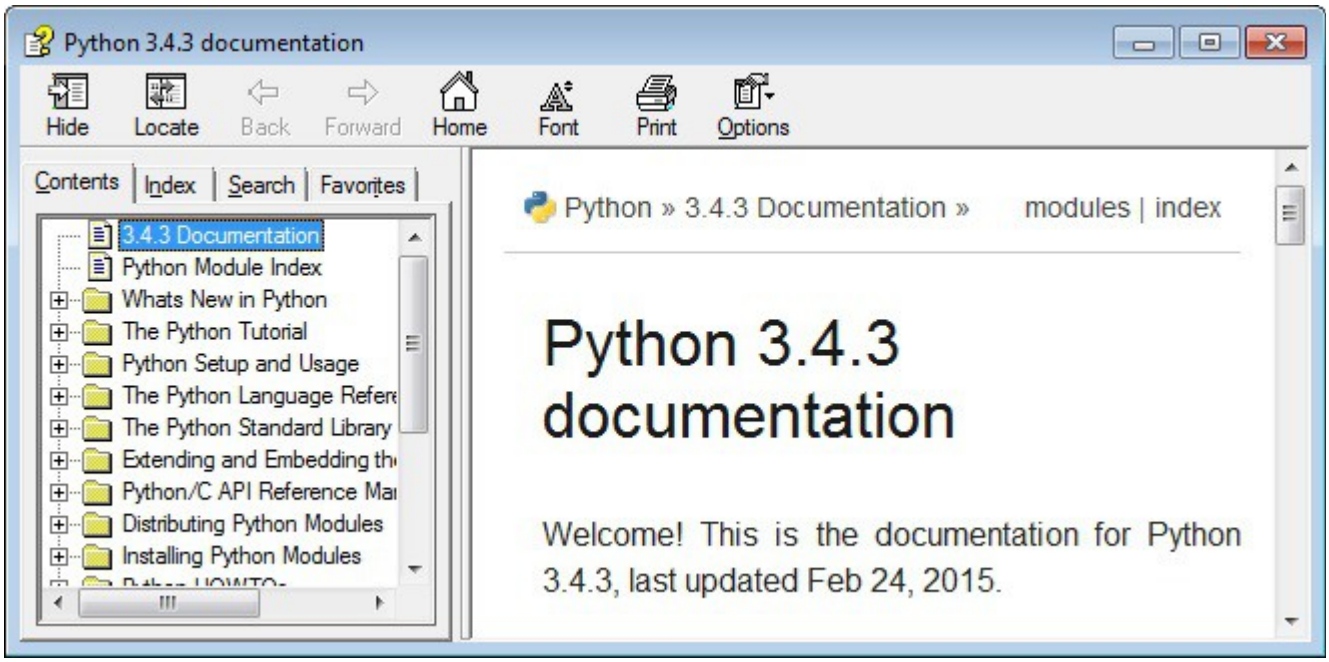
<~> As said, this Application Programming Interfaces (APIs) documentation is related to the very Eric development and, as such, probably more of interest for an Eric's development team rather than for the generality of its users, who here would probably appreciate more a plain documentation of use[102], instead of development.

– –

102<~> For instance, as we dare suggest, an ordered and searchable collection of all Eric's "`What's This?`" hints.

## Help > **Python 3 Documentation**
## Help > **Python 2 Documentation**

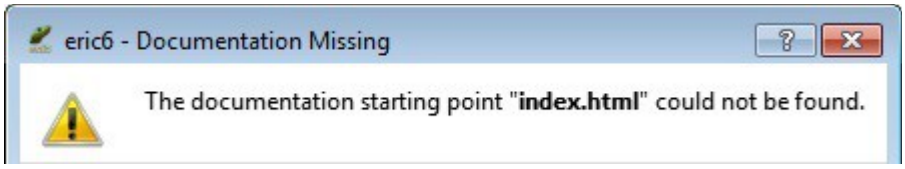
Designed to gain access to the standard Python 3 and Python 2 "`*.chm`" documentation[103], directly within Eric[104].




### *Remark*

That is, provided that such documentation is locally available and the appropriated paths, such as: `C:\Python3X\Doc\python343.chm` and `C:\Python3X\Doc\python271.chm`, had been assigned on: `Settings > Preferences... – Help > Help Documentation, Python Documentation` [*see also:* Eric Setup Completion, *in:* {6.Trail}].

Otherwise all you'll get is such sort of warning:




– –

103HTML Help format file, here viewed by a dedicated Microsoft program.
104Note, once more, that Eric 6 is an IDE designed both for Python 3 and 2.

## Help > Qt4 Documentation

## Help > Qt5 Documentation

Designed to offer access to the original Qt4 and Qt5 documentation directly within Eric, if available. That intended as a source of information complementary to the `Help > PyQt5 Documentation` as hereafter considered [*see*].

### *Remark*

- Qt[105] is the cross-platform GUI toolkit at the origin of the of the very PyQt Python binding, that is the actual package here considered as the Eric prerequisite [*see:* Prerequisites, *in:* {6.Trail}]. As such, it has got a documentation of its own[106] which, nevertheless, is not available within the very PyQt package as here considered [*see: 2/2]* PyQt, *in* {6.Trail}]; that's why here we'll simply mention it, without offering any further operative detail.
- Further reference to this subject can be found on the Digia[107] (http://www.digia.com) and Qt Digia (http://qt.digia.com) web sites. Web references that could be possibly set as direct online links for these `Help` commands, if wanted [*see:* `Settings > Preferences... – Help Documentation, Qt4 / Qt5 Documentation`].

– –

## Help > PyQt5 Documentation

Designed to gain access to the usual PyQt documentation[108] directly within Eric. More precisely, designed to open the Eric Web Browser on the "PyQt Reference Guide" [*see*].

### *Remark*

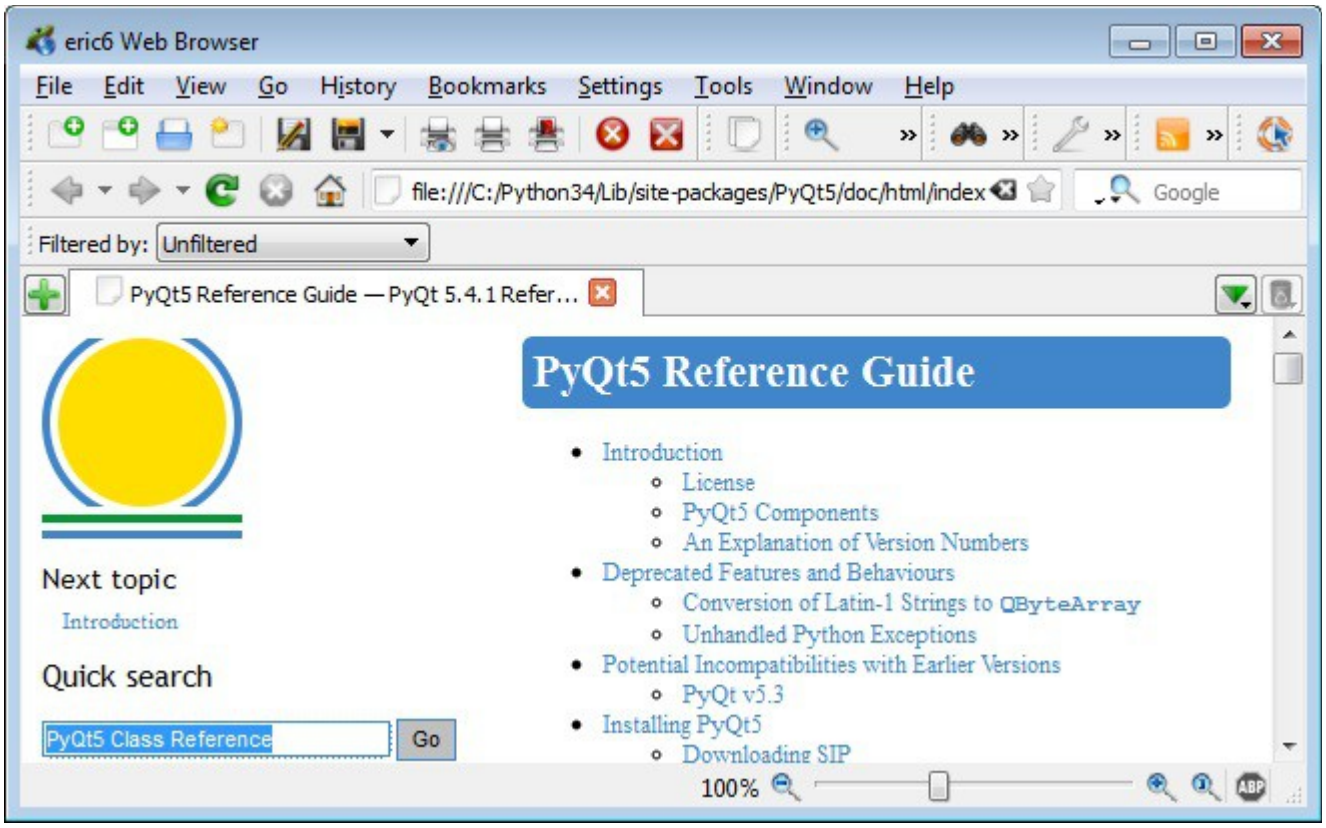
That is provided that, once the PyQt toolkit has been installed, such an appropriated paths as: `C:\Python3X\Lib\site-packages\PyQt5\doc\html\index.html`, had been assigned on: `Settings > Preferences... – Help > Help Documentation, PyQt5 Documentation` [*see:* Eric Setup Completion, *in* {6.Trail}].

105<~> We heard that "*Qt*" stands for a self-praising "*cute*" …

106Whose role and utility is unknown to us, we mean: beyond the specific PyQt documentation.

107Digia is the Finnish software company that has acquired Qt from Nokia in year 2012.

108Available as part of such standard "`PyQt GPL v5.4.1 for Python v3.4`" installation kit, in form of two PyQt Documentation items: "Reference Guide" and "Class Reference".




Note that on this "PyQt Reference Guide" contents directory you'll find also a useful link to the "PyQt Class Reference" [*see*].

### *Viewpoint*

Note that this PyQt Reference is a typical documentation for consultation only, that is rather unpractical for newcomers. For friendlier source of info see section Reference Book List [*in:* Appendix].

– –

## Help > **About Eric6**

[109]Designed to show this product's identity card, comprising: official denomination, revision code, authors, acknowledgments and License (GNU – General Public License).

109Actually, a Core Plugin item [*cf.:* `Plugins > Plugin Infos...`].


And comprising also two most appreciable e-mail references, where users are invited to express their related problems and wishes[110]:

```
eric-bugs@eric-ide.python-projects.org
eric-featurerequest@eric-ide.python-projects.org
```

– –

### *Viewpoint*

Throughout this Report of the different ways, as here shown, to write this very product's name, such as: ***Eric***, ***eric6***, ***ERIC***, for reasons of uniformity we have adopted this one:

**Eric**

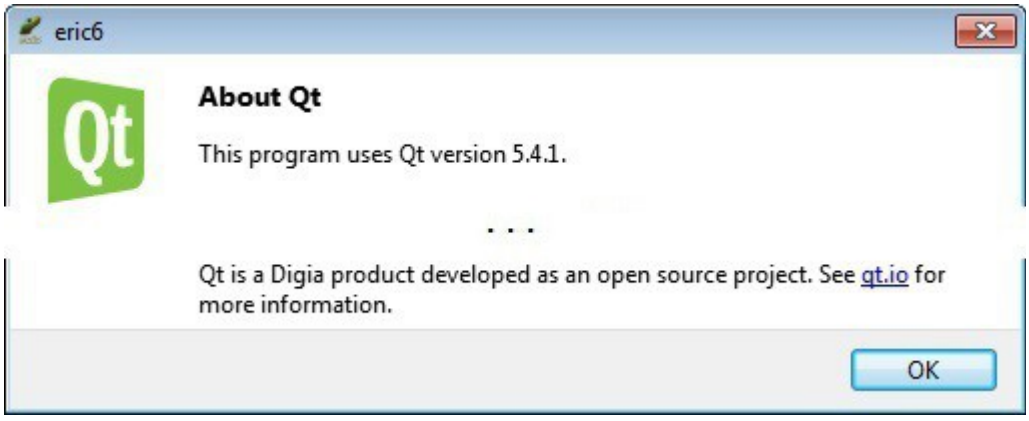
with initial uppercase and, possibly, with the final rev. number "**6**" only when assumed as appropriate to be so specific.

– –

## Help > **About Qt**

Designed to show a copyright acknowledgment box for the Qt GUI toolkit, as here utilized.

110<~> By the way, for matters concerning this very Report instead, you may refer to: Studio-PM@hotmail.com

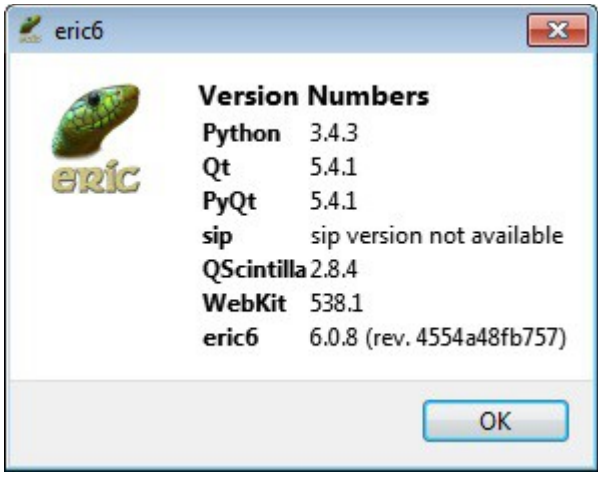



More precisely, as this Qt GUI toolkit is here utilized, that is through its PyQt Python binding [*see:* Prerequisites, *in:* {6.Trail}].

– –

## Help > Show Versions

Designed to show a box listing all main items currently integrated into this unique IDE environment, along with their precise brand names and version codes [*cf.:* Prerequisites, *in:* {6.Trail}, *and also:* `Settings > Show External Tools`].

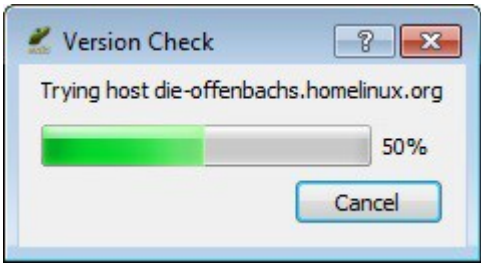



– –

## Help > Check for Updates...

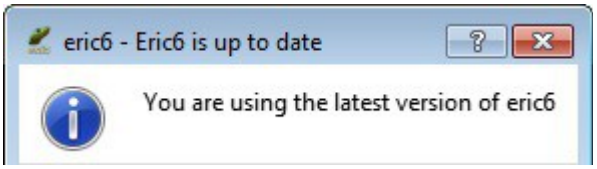
Designed to activate an automatic on-line comparison between the local version of the Eric product

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, and possibly get it.




To execute, this command requires an active Internet connection. Here is a possible result:


Or:




For more information about the Eric versions available for down-loading, see next command: `Help > Show Downloadable Versions...`

### *Remark*

The updates here considered are all related to the same Eric main ver. 6, that is: not to possibly other Eric main versions, being considered altogether as a different products[111].

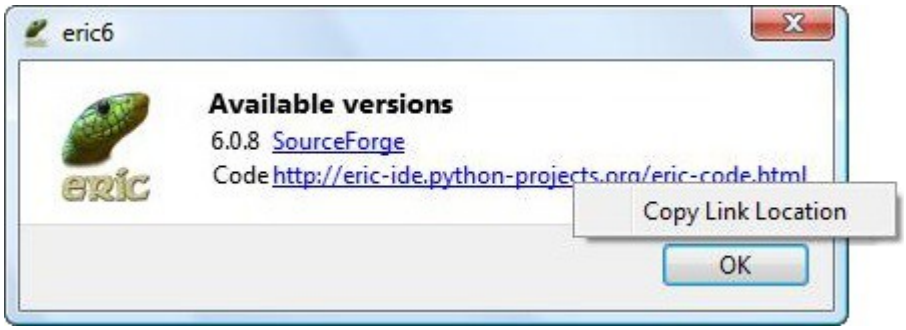
– –

## Help > **Show Downloadable Versions...**

Designed to display some more info—besides what offered by `Help > Check for Updates...` [*see*]—about the possibly different Eric versions currently available for downloading. To execute, this

111A useful Remark at the time of Eric's version 4 and 5, concurrently kept for the two distinct Python ver. 2 and 3. Now, with Eric 6, not so significant [*ref. to:* Compatibility with Python ver. 3 or/and 2, *in* {0.Lead} Essentials].

command requires an active Internet connection.




Of the two links in display, the first one points to the so declared last "stable" version, the second to the the most recent source version currently under development.

### *Remark*

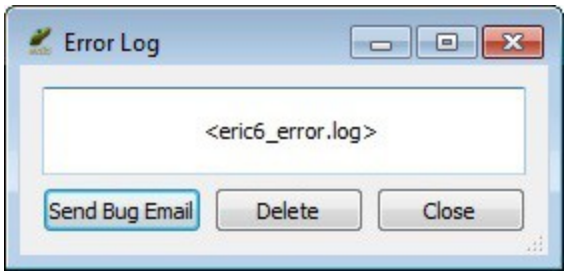
Given a look at these links, they currently revealed as:

http://sourceforge.net/projects/eric-ide/files/eric6/stable/6.0.8
http://eric-ide.python-projects.org/eric-code.html

– –

## Help > **Show Error Log...**

Designed to show on an "`Error Log`" dialog box the last log file possibly generated in case of an Eric's execution error condition. Command disabled when no such file available.




Error log file, automatically recorded and stored into the "`<User>\_eric6`" directory, that is on a per-User base, under the name "`eric6_error.log`". It could be also used to document the error event to the Eric's producer [*cf. also: next command* `Help > Report Bug...`].

– –

## Help > **Report Bug...**

## Help > **Request Feature...**

Two commands aimed at granting the users a channel of dialog with the Eric producer. Both require an adequate `Settings > Preferences... – Email, Configure Email`[112].

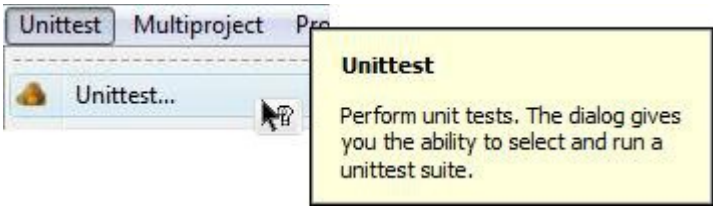
### *Viewpoint*

A highly appreciable attitude, confirmed also by the e-references of dialog offered in a much simpler and direct way, with the `Help > About Eric6` command [*see*].

– –

## Help > **What's This?**

Designed to show a short description on a pop-up help box, just when pointing at any given control item on the Eric window.




Here the corresponding "`Shift+F1`" function key shortcut reveals particularly handy.

### *Viewpoint*

A mechanism not always so practically useful as it could be. In fact, as in this real example, suppose we have no idea what a "`Unittest`" command is, therefore we ask: "`Help > What's This?`"

As a result we get a hint box telling us that the `Unitest...` command is there to:

"*`Perform unit tests`*"

Which is an assertion so obvious to be completely useless. And, please, don't tell us that a Mr. User should already know what a `Unittest` is, 'cause, by this token, the best policy would not to offer any hint message at all. Plus this is not fair, as it's precisely when you don't know that you ask, isn't it?

What we mean, here for instance, is that in such a case it would suffice just to recall and clarify

112<~> A not so immediate setting, we dare say, as are the "`Outgoing mail server (SMTP)`" and the "`Outgoing mail server port`" parameters [*see*].

something about the "*standard Python unittest module*", so to be of real help to the poor Mr. User. Who, this way, could be pointed to the right direction where to go so to learn more, would he wished to.

-<>-

# Tool Bars

Default Tool Bars, on the "North Side" of the Eric Application Window, located below the Main Menu Bar:


and at the right side of the same window:

Custom configurable [*see:* `Window > Toolbars`, *and* `Settings > Toolbars...`], and independently for each one of the two Profiles of use, Edit / Debug [*see:* `Settings > View Profiles...`].

– –

A rich set of icon-tools designed to constitute an efficiency aid for calling the menu commands most frequently used either in Edit or Debug Profile of use. Role of each icon is hinted by a tool-tip, such as the "`Open (Ctrl+O)`" here shown in the above figure, reporting both the corresponding menu command's name[113] and its short-cut.

Functional description of each icon's command can be found on the dedicated section Main Menu Bar, and Commands [*top section here in:* {1.North}].

– = –

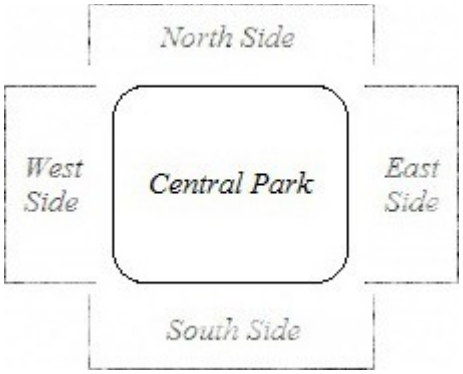
113So that you may refer to the related Report section for a complete functional description of the named command.

# {2.Central} Eric Window – *Central Park*

*S-PM 160400*

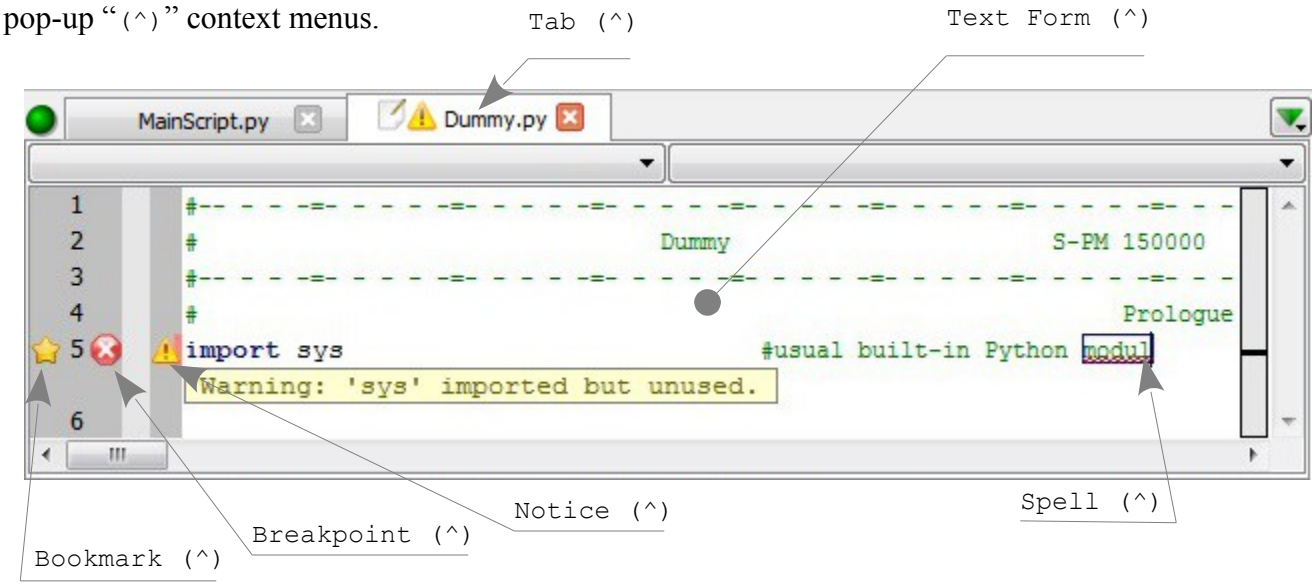
## Text Edit Central Pane

Here we are at the "Central Park" area of the Eric Application Window [*cf.:* Eric Window Map*, in:* {Map}*, at the end of this* Report].




In essence it's a stable pane—that is, not optional [*cf.: next "*Auxiliary ... Pane*" sec.s, in:* {3.South}]—positioned at the center of the Eric window's work area, aimed at hosting a set of tabbed forms where to edit Python source files. Hosting also a main context menu, along with some other contour context menus, and a few other related controls, as hereafter listed [*see: next* Text Editing Usability *sec.*].

<。> Here see a typical snapshot of such pane, showing also where to locate and identify the available pop-up "(^)" context menus.

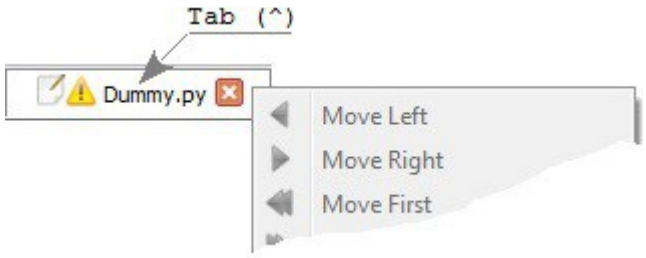



– –

### *Context Menu List*

`Tab (^)` `Text Form (^)` `Bookmark (^)` `Breakpoint (^)` `Notice (^)` `Spell (^)`

A detailed description of all the pop-up context menus available on this Text Edit Central Pane, from "`Tab (^)`"




to "`Spell (^)`" [*ref. above* Context Menu List], will be then subject of the subsequent sections.

– –

## Pop-up Context Menus – Generalities

<!> The above shown is just one example of the many, really many, Context Pop-up Menus that enrich Eric's usability, as hereafter described in this Report. To activate any such a pop-up menu:

- Point the mouse on the context zone—that is, in this case: at the very Tab-label—, then right-click. An action here represented throughout this Report with a symbolic: "`(^)`"
- As alternative action, valid only on the source text area: (`Shift+F10`), or Application key [*cf.: next* Text Form (^) Menu *sec.*].

### *Viewpoint*

<.> Eric context menus may comprise both original context-dependent commands and also some of the main menu commands, duplicated and available as context commands just for a shear reason of convenience.

<!> In this Report, as a rule, only the original context-dependent commands will be here on described. Whereas, about the duplicated commands, just a reference of the corresponding main menu will be offered; e.g.: [*ref.:* `File >`] `Close` with no duplicated description.

<!!> *Be also aware that the context version of some main menu commands may slightly differ from their main menu's model, typically just in their scope, possibly limited to the referred context. For manifest reason of practicality also such context-duplicated commands will be here not specifically treated, but referred to their main menu version, with their peculiarities confidently left to the adventurous user's intuition.*

– –

# Text Editing Usability

Friendliness of usage of the Eric source text edit form is assured by some tools and features well worth to be known in some detail.

– –

## *Text Editor*

To the Eric *Text Form* is associated a powerful *Text Editor*, so intuitive that it doesn't require any explanation, but just an introduction of some of the associated text editing aids, along with the description of all associated controls [*see: next sec.s*].

– –

## *Text Edit Aids*

Some notable Text Edit Aids, provided to support the source text editing activity.

Syntax Highlighting

To visually enhance the readability of the texts to be edited. As, for instance, in such a line: `import sys  #-standard Python library`,
where you distinguish a blue keyword, a black identifier and a green comment.

A feature operating according to each possibly different coding language, say: Python, Ruby, XML, …; and based upon such lexical analyzer—i.e.: "lexer"—as QScintilla, as included in the PyQt package [*cf.:* Prerequisites, *in:* {6.Trail}].

Which the coding language currently in use will be derived from the related source file's extension, as matched via the Preference setting [*see: commands* `Settings > Preferences... – Editor > Highlighters > Filetype Associations` *and* `Project > Lexer Associations...`].

Coding Aids — Text Autocomplete and Calltip features, as with the commands comprised into the sub-menu `Edit > Complete` [*see*].

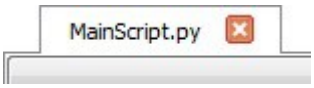
Spell-checker — Special feature requiring the presence of the optional PyEnchant toolkit [*see:* `Extras > Check Spelling...`].

– –

## *Central Pane Control List*

Green *Pin Marker*, to tell which is the currently active Tab; red colored  for the not-active Tabs. Particularly relevant in case of split views [*ref.:* `View > Split View` *command*].

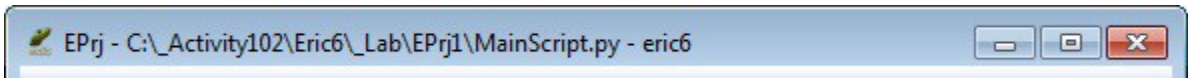
– –



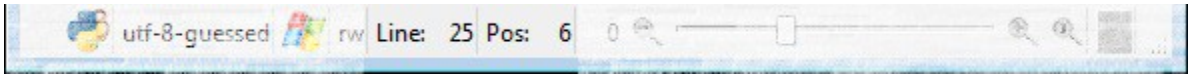

*Tab-label*, for the Text Form selection, with a "**x**"–`Close` button.

## *Remark*

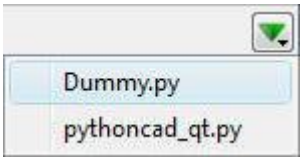
Note that on the Title Bar, at the top of the Eric Application Window [*ref.:* {Map} *trailer sec.*], you'll see displayed the full path of the currently active source file




and, on the Information Bar, at the bottom of the same window, you'll see the `Line` and character `Position` of the current text insertion point.




– –

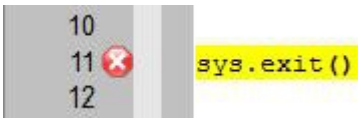



*Navigation List*, at the top-right corner of the text edit pane. Click the down-pointer  to show.

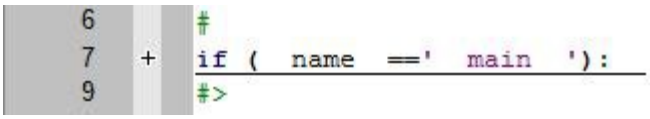
– –


*Vertical Ruler Bar*, to display text line numbers and, possibly, the icon marks of the following toggle controls, in left-right order: "Star" Bookmark, "Cross" Breakpoint, "+" Expand / "–" Collapse Fold, and Notice [*see hereafter*].


"Star" *Bookmark*, with associated context menu [*ref. next "(^)" sec.*].




"Cross" *Breakpoint*, with associated context menu [*ref. next "(^)" sec.*].

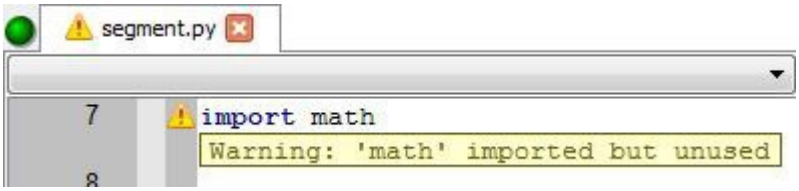



"+" *Expand* / "–" *Collapse Fold* of source blocks.

## *Remark*

Actual style of this fold column can be chosen with the "`Folding style`" drop down list, on the "`Margins`" area of the "`Configure editor styles`" [*see:* `Settings > Preferences... – Editor > Style`]. What's here shown is the default "`Plain`" style [*default, indeed: as usual for this* Report].

– –




e.g.:

*Notice Icon*, and message;

a "⚠"–Warning.

## *Remark*

Here you see that the "⚠"–Warning icon is duplicated on the very Tab-label, so to inform you that somewhere, possibly far away from the source section in sight, there is such a warning condition, probably worth your attention. It is in such a case that this quick-jump command comes rather handy: `Bookmarks > Next / Previous Warning Message` [*see*].

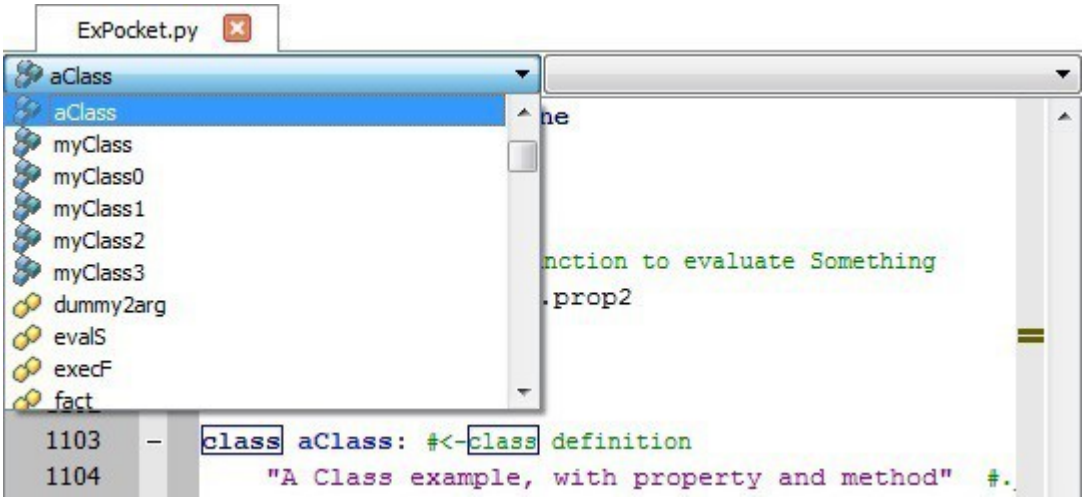
– –

## *Drop-down Object Lists*

A pair of drop-down lists, designed to see and navigate amongst the objects of the current source module.

### *A first List,*

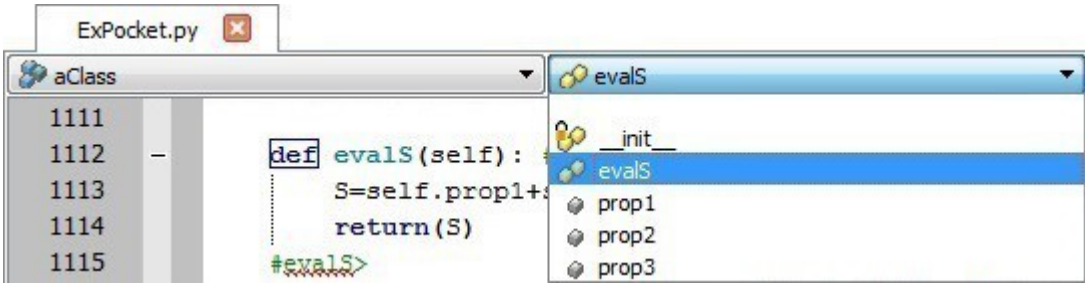
at left, is to show all classes and functions.




– –

### *A second List,*

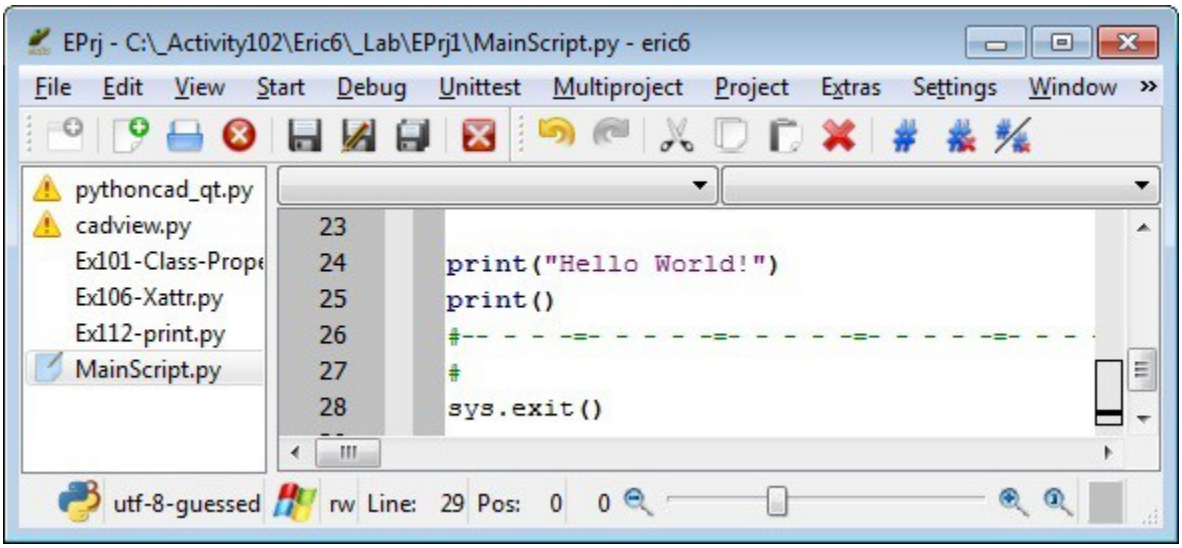
at right, is to show the methods and attributes of the class possibly selected on the first list, at left [*as in this "*`aClass`*" example*].




– –

## *Alternative Text "List View" Form*

A text "List View" form can be activated as an alternative to the usual tabbed view form, as so far always here shown [*cf. following example, with the left-list of items to select, instead of tabs*].




To that purpose, with command `Settings > Preference... - Interface > Viewmanager`, on the `Configure viewmanager` control form[114], you may select the `Window view: Listspace` [*see*]. That is, as an alternative to the initial `Tabbed View` default condition. Such new condition will be then enabled—or reset—at the next application startup.

## *Viewpoint*

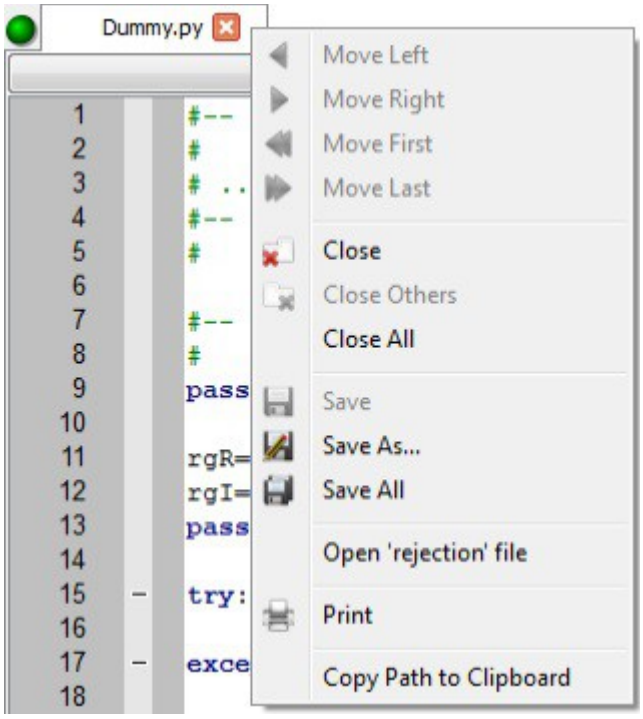
<。> As with other similar cases, throughout this Report we'll make reference only to the initial default configuration, leaving the adoption of a possible alternative to the free preference of the adventurous user.

-<>-

114Actually, two Core Plugin `VmListspace` / `VmTabview` items [*cf.:* `Plugins > Plugin Infos...`].

## Tab (^) Menu

A right-click "(^)" on the very Tag-label, and you'll get the following pop-up context menu.




### *Command List*

| | | | | |
|---|---|---|---|---|
| | Move Left | Move Right | Move First | Move Last |
| [*ref.:* | File >] Close | Close Others | Close All | |
| [*ref.:* | File >] Save | Save As... | Save All | |
| | Open 'Rejection' File | | | |
| [*ref.:* | File >] Print | | | |
| | Copy Path to Clipboard | | | |

– –

### *Remark*

<。> Note that, as a rule for all context menus treated in this Report, in the above Command List such a reference to the corresponding main menu command: [*ref.:* File >] Close is to say that the related description will be here not duplicated [*cf.:* *above* Pop-up Context Menus – Generalities *sec.*].

– –

### Tab (^) **Move Left**
### Tab (^) **Move Right**
### Tab (^) **Move First**
### Tab (^) **Move Last**

Designed to move the currently active Tabbed Text Form to the named position, if within a set of more than one such forms; commands otherwise disabled. Same action can be done through the drag-and-drop action on a Tab-label.

– –

### Tab (^) **Open 'Rejection' File**

Designed to open the *Rejection* file possibly associated to the currently active source text file. That is a file located in the same directory and named the same way of current source text file, plus a "`*.rej`" extension appended, and typically generated automatically as consequence of an unsuccessful VCS merging process [*see: main command* `Project > Version Control`].

This context command is disabled when no such Rejection file results available.

– –

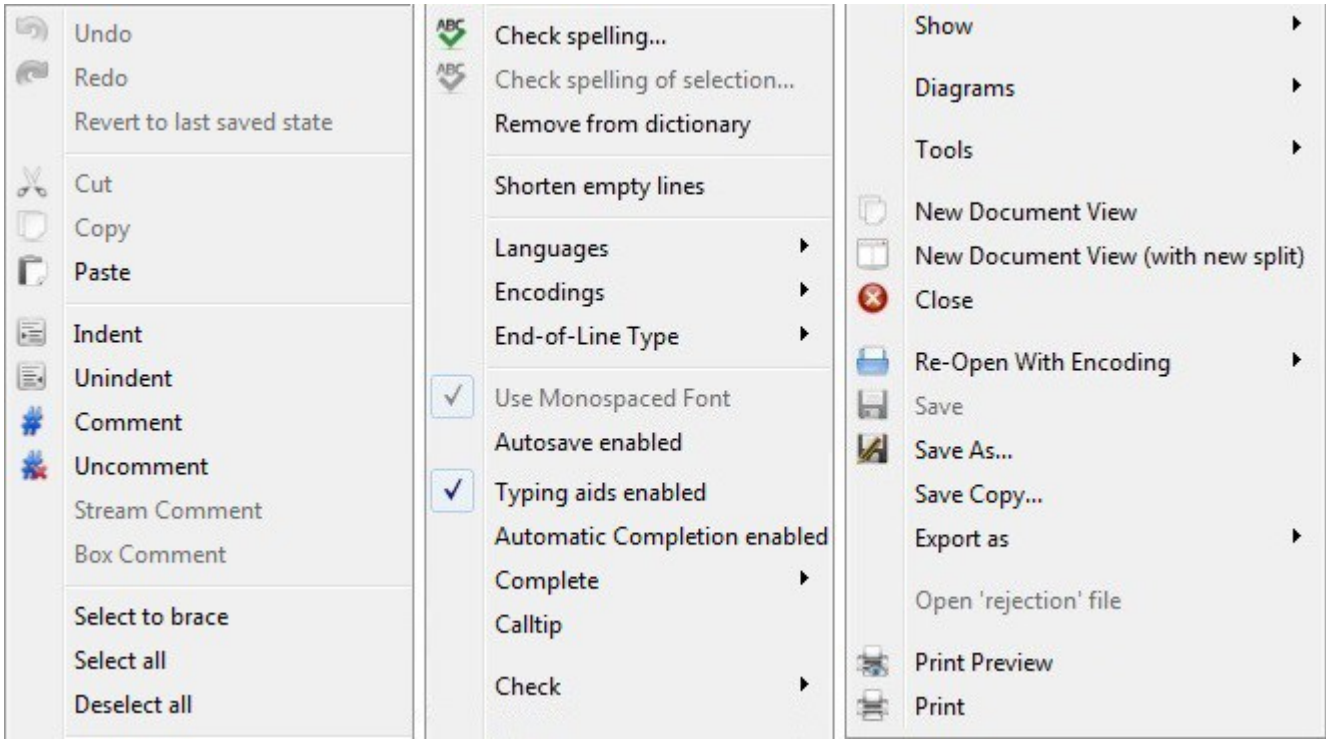
### Tab (^) **Copy Path to Clipboard**

Designed to copy into the system clipboard the currently selected item's path [*cf.: above* Central Pane Control List *sec., first* Remark], possibly ready to be then "pasted" somewhere. A little handy feature useful for when such an argument is then required, and no search button is in sight.

–<>–

# Text Form (^) Menu

Normal pop-up context menu [*vs. "*reduced*", cf.: next* Remark], as it appears right-clicking on the Text Edit Form.




## *Remark*

A reduced version of this context menu can be obtained checking the "`Show minimal context menu`" control box[115] on the `Settings` > `Preferences... - Editor > Style, Configure editor styles` control form [*see*].

– –

## *Command List*

| | | | |
|---|---|---|---|
| [*ref.:* `Edit >`] `Undo` | `Redo` | `Revert to Last Saved State` | |
| [*ref.:* `Edit >`] `Cut` | `Copy` | `Paste` | `Indent` |
| [*ref.:* `Edit >`] `Unindent` | `Comment` | `Uncomment` | `Stream Comment` |
| [*ref.:* `Edit >`] `Box Comment` | `Select to Brace` | `Select All` | `Deselect All` |

115 But in this case too, as it is usual in this Report, we'll make reference to the initial default condition only, leaving other possible alternative configurations to the initiative, and preference, of the adventurous user.

```
[ref.: Extras >]  Check Spelling...
     Check Spelling of Selection...  Remove from Dictionary
[ref.: Edit >]    Shorten Empty Lines
     Languages  Encodings     End-Of-Line Type      Use Monospaced Font
     Autosave Enabled         Typing Aids Enabled   Automatic Completion Enabled
[ref.: Edit >]    Complete[116]    Calltip
[ref.: Project >] Check

     Show
[ref.: Project >] Diagrams[117]
     Tools
[ref.: View >]    New Document View      New Document View (with New Split)
[ref.: File >]    Close
     Re-Open with Encoding
[ref.: File >]    Save                  Save As...   Save Copy...   Export As
[ref.: Tab (^)]   Open 'Rejection' File
[ref.: File >]    Print Preview         Print
```

### *Remark*

Description of context commands identical or equivalent to those that can be found in the main Menu Bar [*see in:* {1.North}] will be not hereafter repeated. Also possible minor differences, due to the context condition, will be not treated, and left to the active user's interpretation.

– –

## Text Form (^) **Check Spelling of Selection...**

Designed to operate exactly as the main menu command `Extras > Check Spelling...` [*see*], but on a selected section of text.

– –

116 [*Eric Feature Under Revision*] A minor discrepancy here about "`dynamic`" sub-command label [*see*], scheduled to be fixed next Eric ver. 6.2.

117 For some reason this context sub-menu is richer the the referred main menu correspondent, as it comprises these other sub-commands: `Class / Package / Imports Diagrams...` [*see*]. Anyhow we still consider this as a specialized topic, out of this Report's scope, and about which we haven't anything here to add to what already said.

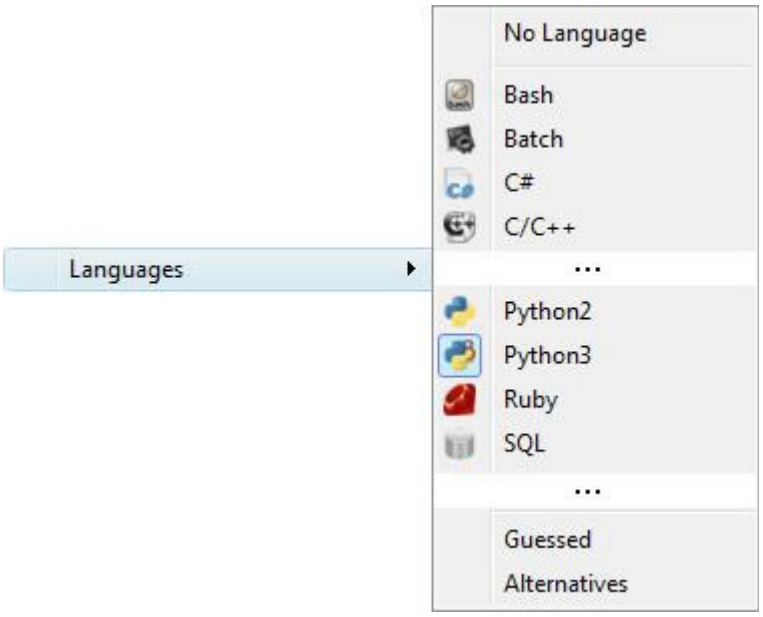
## Text Form (^) **Remove from Dictionary**

Designed to do the inverse action of the context command `Spell (^) Add to Dictionary` [*see:* Check Spell Menu *sec.*] for a word selected on the source text.

– –

## Text Form (^) **Languages**

Designed to show such a rich list of languages:




where to possibly change which the related Text Edit Aids to enable for the current session, typically should the automatic "guess"—that is: detection—fail [*cf.:* How to Select the Language Interpreter, *in:* {0.Lead}].

The currently enabled language is marked with an enhanced icon on this list, and also with the same icon shown at left corner on the Information Bar, at the bottom of the Eric Window [*cf. in:* {Map}]. E.g., for Python 3:

<。> Note that with a click on this very icon you'll get this same `(^) Languages` pop-up command.

### *Sub-Command List*

`No Language` `Guessed` `Alternatives`

`> No Language` No coding language text aids enabled, just a plain text spell check [*cf.:* `Extras > Check Spelling...`];

`> Guessed` Automatic detection of the coding language;

`> Alternatives`
An "`Alternative Lexer`" language choice, as with `Project > Lexer Associations...` [*see*].

### *Remark*

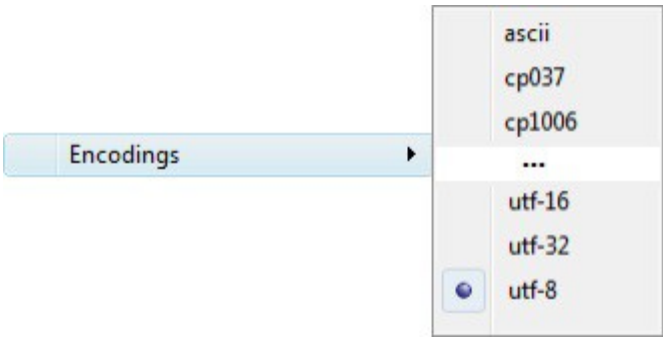
<。> Note that this language selection doesn't concern the source text execution, but exclusively its editing, and for such aspects as:

- Syntax highlighting;
- Autocomplete and Calltip;
- Text Typing Aids, that is spelling check and display;
- File type setting, and related.

– –

## Text Form (^) **Encodings**

Designed to show a list where to possibly select which the Unicode character encoding code to enable for the current text edit session [*see also:* How to Select the Unicode Encoding, *in:* {0.Lead}].




The currently selected code is marked with a blue head-pin on the list [*see*] and, when changed, will

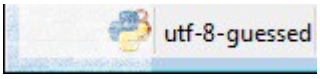
take effect for next conversion on. Context command `(^) Re-Open with Encoding` is provided to re-interpret the already opened source text with a possibly changed code [*see*]. Currently active code is signaled externally at left on the Information Bar, at the bottom of the Eric Window, besides the icon of the current editing aids language [*cf. in:* {Map}]. E.g.:


<。> Note that with a click on this very icon you'll get this same `(^) Encodings` pop-up command.

## *Remark*

See hereafter some specific notes about this subject, moreover assumed in general terms as well known to Python programmers.

- Unicode Encoding is relevant when working with strings, therefore is relevant also at the very opening of a Python source module, before its execution, being it read as Python strings. Python keywords and identifiers, being anyhow restricted to the ASCII character set, are not affected by this encoding [*cf. next point*].
- Python strings may be defined as *byte strings* or as *Unicode strings*. Python, if not differently instructed, when needed will use your computer's default "locale" character set to convert byte strings into Unicode character strings; locale that, of course, may differ in different systems.
  A standard way to explicitly instruct Python how to execute this conversion is to use the so called "magic comment" [*e.g.:* `# -*- coding: UTF-8 -*-` ], as here hinted at the section How to Select the Unicode Encoding [*see in:* {0.Lead} *Essentials*]. Magic comment will take effect from the very opening of a module, included. The here considered context command can take effect only afterwords.
- With no initial encoding's "magic comment", nor explicit `(^) Encodings` actions, an auto-detection mechanism will take control, typically leading to a "`utf-8-guessed`" condition [*see*].

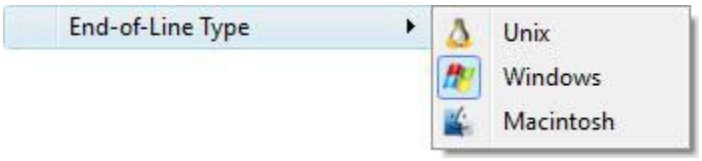



- The here considered context command is meant to take control and possibly change the current Unicode encoding set used by Python to convert binary byte values into Unicode characters. For instance "`UTF-8`" is a common Latin Unicode encoding, "`koi8-u`" a Cyrillic one.
- "`invalid or missing encoding declaration`" is the typical error log message should Eric detect any encoding error conditions.

– –

## Text Form (^) **End-Of-Line Type**

Designed to show such a list:




where to possibly change which the type of End-Of-Line to enable for the current text edit session[118] [*see also:* `Settings > Preferences... – Editor > Filehandling, End of Line Characters` *and:* `Edit > Convert Line End Characters`].

Also the currently enabled EOL type is shown on the Information Bar [*cf.: preceding commands*]. E.g.:


<。> Note that with a click on this very icon you'll get this same `(^) End-Of-Line Type` pop-up command.

– –

## Text Form (^) **Use Monospaced Font**

Designed to possibly toggle between monospaced / proportionally spaced typeface fonts, for the current text edit session [*see also:* `Settings > Preferences... – Editor > Style, Configure editor styles, Fonts`].

– –

## Text Form (^) **Autosave Enabled**

Designed to possibly enable / disable the timed autosave action occurring at the "`Autosave interval`" time, as set on the "`Configure file handling settings`", "`Save`" area [*see:* `Settings > Preferences... – Editor > Filehandling`].

– –

118Current text edit session only. As a rule, all Eric context commands are tied to the current context scope only.

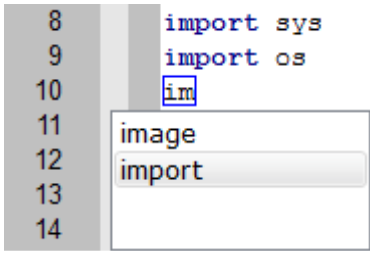
## Text Form (^) **Typing Aids Enabled**

Designed to possibly enable / disable the "*Typing Aids*" for the current text edit session, as they have been defined with command `Settings > Preferences... – Editor > Typing, Configure Typing`, for the language Python (2 and 3) or Ruby [*see*].

– –

## Text Form (^) **Automatic Completion Enabled**

Designed to possibly enable / disable such automatic completion feature as hereafter exemplified:

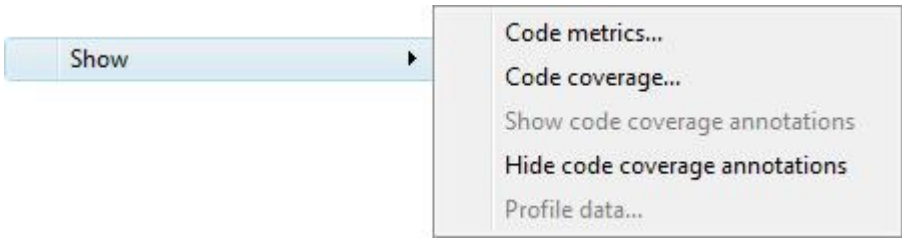



during the current text edit session, only. To permanently set such condition, see the `Automatic Completion Enabled` check-box, on `Settings > Preferences... – Editor > Autocompletion` [*cf. also:* `Edit > Complete` *command*].

– –

## Text Form (^) **Show**

Designed to show this box of sub-menu commands, aimed at controlling the display of specific information; as hereafter detailed.



### *Sub-Command List*

[*ref.:* `Project > Show`] `Code Metrics...` `Code Coverage...`
`Show / Hide Code Coverage Annotations`
[*ref.:* `Project > Show`] `Profile Data...`

– –

## Text Form (^) Show > **Show Code Coverage Annotations**
## Text Form (^) Show > **Hide Code Coverage Annotations**

Designed to show / hide the Coverage marks located on the rightmost column of the vertical ruler, according to the currently recorded result of a `Start > Coverage Run` command execution [*see*].

```
14  –  try:
15         rgR=rgR/0
16         print("Result: ", rgR)
17  –  except:
18         rgR=0.1
19         print("except")
20     #>
```

In the above example, for instance, you see a statement flagged as not executed because of an intercepted exception.

– –

### *Viewpoint*

<~> As already noted elsewhere [*e.g. with:* `Debug > Toggle Breakpoint` *and* `Start > Profile Script...`], here too we've realized that the treatment of the Python "`pass`" statements is rather peculiar. Indeed, as you can see in the following example:

```
12     rgI=0                #Integer    0   1
13  –  try:
14         rgR=rgR/rgI
15         rgI=1            #Integer
16         pass
17         print("Result: ", rgR)
18         pass
19  –  except:
20         rgR=0.1
```

a "`pass`" statement (as here at line 16) may not be red-flagged as any other statements when not-covered, even though correctly declared as not-executed onto the related `Text Form (^) Show > Code Coverage...` lists [*verify*].

<~>That is, as if a `pass` statement, sometimes, but not always (as then, for instance, here at line 18) could be treated as it were a comment, or an empty line [*well, we do not agree*].

– –

## Text Form (^) **Tools**

Designed to list and run one of the so called *Plugin* Tools currently installed, if any [*cf.:* `Plugin > Install`]; command disabled otherwise.

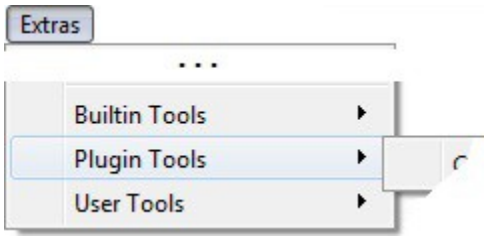

The above not-empty list here shown is just an example, therefore with no related functional description then offered.

<。> About the Plugin subject in general and, specifically, how to install and manage these *Plugin* Tools you may refer to the Plugins Command Menu section [*see in:* {1.North}].

– –

### *Remark*

Eric Tools are organized into three distinct set-containers, called *Tool Groups*: the two initial standard *Builtin* and *Plugin* Tools, and then the *User* ones, possibly added subsequently; as shown in the corresponding `Extra` main menu command [*see*], hereafter reproduced.



<。> Note that this local `(^) Tools` context command has been conveniently designed to offer you directly the so called Plugin Tools[119], in that corresponding exactly to the second one of these three main menu's companion commands `Extras > Builtin / Plugin / User Tools` [*see in:* {1.North}].

– –

## Text Form (^) **Re-Open with Encoding**

Designed to re-read and then show the current Python source module making use of a possibly different Unicode encoding, as here selected [*cf. also:* `(^) Encodings`]. This command will be necessarily disabled if any conflict—that is: any difference—between the current text as in display and the original version as saved on disc. About which is the encoding currently enabled, see the Information Bar; e.g.:


Just for example, consider how the same Python comment will appear when re-read with the "wrong" Unicode encoding "`koi8-u`": `# Gru╪u÷en` , instead of with the "right" encoding "`utf-8`": `# Grüßen`

<。> Possibly different encoding set here will take precedence over such initial "magic comment", as:

```
# -*- coding: UTF-8 -*-
```

enabling a possibly different encoding [*cf.:* How to Select the Unicode Encoding, *in:* {0.Lead} *Essentials*]. Thus note also that the source Python module as originally read into Eric could possibly be different from the same module as then read into different environments, under different encoding conditions.

– –

### *Warning*

Be aware that saving a wrongly–interpreted source text may lead to its permanent corruption. That is to say:

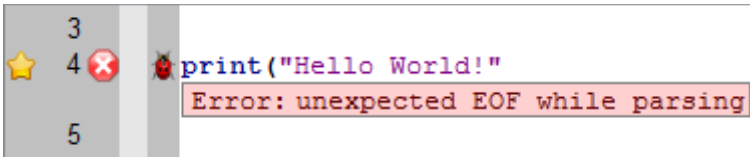
<!> *Before saving a source text with a changed encoding, think twice (and keep a back-up copy)*.

–<>–

119<~> "conveniently" might it be, but surely a bit confusingly too; as to say "Tools" instead of "*Plugin* Tools" is like to say New York to signify Manhattan.

## Vertical Ruler, Context Menus

Three distinct pop-up context menus associated to the Vertical Ruler columns,

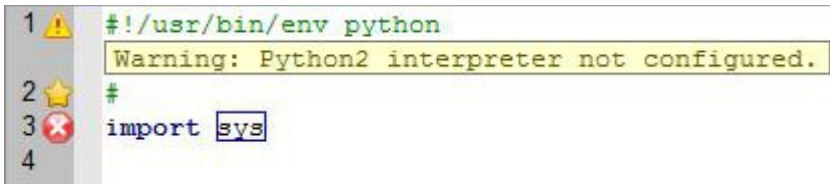



here orderly listed from left to right: `Bookmark`, `Breakpoint` and `Notice`.

With this rightmost `Notice` column for such markers as: Syntax Error, Code Warning, Code Coverage [*see:* `Start > Coverage Run`, *and:* `Text Form (^) Show > Show Code Coverage Annotations`], Extracted Task [*see:* `Bookmarks > Next Task`], and Code Change [*see:* `Bookmarks > Next Change`].

### *Remark*

A more compact configuration of this Vertical Ruler and, consequently, of the related context menu, can be obtained checking the "`Show unified margins`"[120] control box, on the `Settings > Preferences... – Editor > Style, Configure editor styles` control form [*see*].

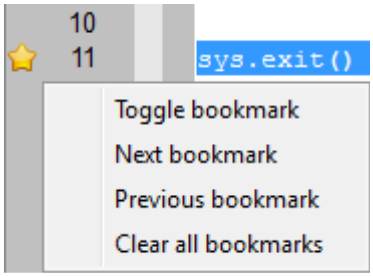



But in this case too, as it is usual in this Report, we'll make reference to the initial default condition only, leaving other possible alternative configurations to the initiative, and preference, of the adventurous user.

-<>-

120With "*margin*" that refers to what here we prefer to call "*column*".

# Bookmark (^) Menu




## *Command List*

[*ref.:* `Bookmarks >`] `Toggle Bookmark` `Next Bookmark` `Previous Bookmark` `Clear All Bookmarks`

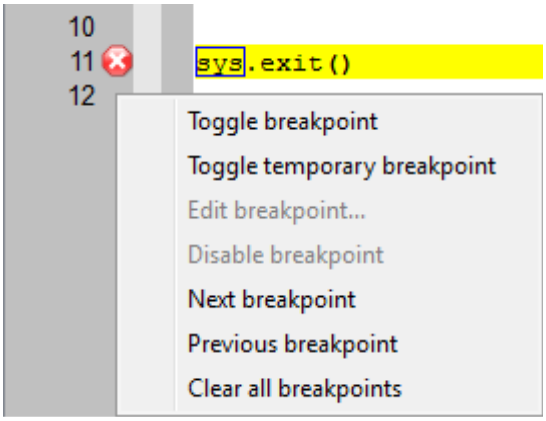
– –

## Bookmark (^) Clear All Bookmarks

To clear all bookmarks defined on the currently active Text Edit Form [*cf.: command* `Bookmarks > Clear Bookmarks`].

– –

# Breakpoint (^) Menu



### *Command List*

[*ref.:* `Debug >`] `Toggle Breakpoint`
`Toggle Temporary Breakpoint`
[*ref.:* `Debug >`] `Edit Breakpoint...`
`Disable Breakpoint`
[*ref.:* `Debug >`] `Next Breakpoint` `Previous Breakpoint`
`Clear All Breakpoints`

– –

## Breakpoint (^) **Toggle Temporary Breakpoint**

To set / clear a breakpoint on the corresponding source line. Almost identical to the main menu command `Debug > Toggle Breakpoint`, but for the automatic setting of the related `Debug > Edit Breakpoint...`'s "`Temporary Breakpoint`" check-box [*see*].

<。> We recall that Temporary Breakpoints are meant to be executed just once, then disabled.

– –

## Breakpoint (^) **Disable Breakpoint**

To disable the currently pointed breakpoint, without clearing it.

– –

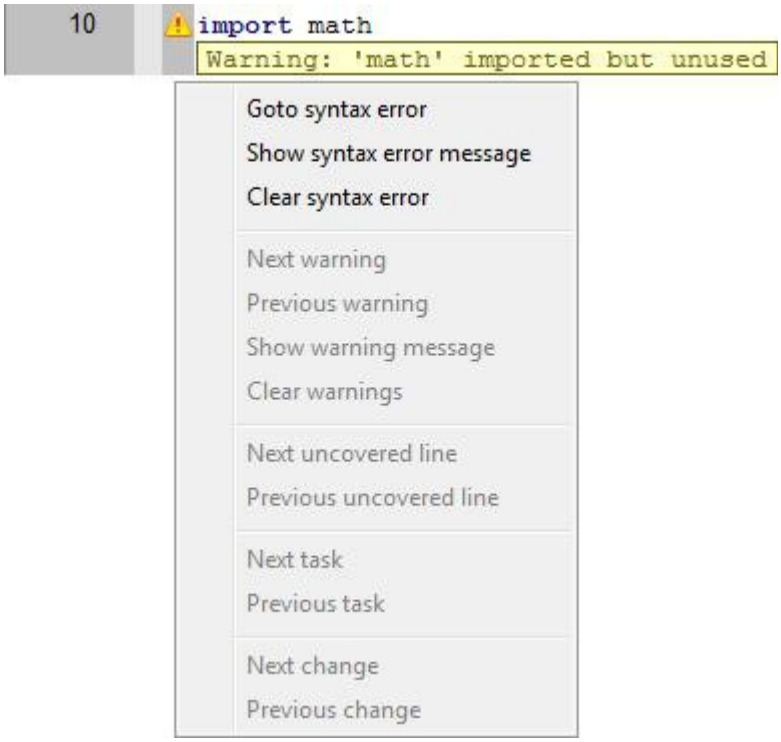
## Breakpoint (^) **Clear All Breakpoints**

To clear all breakpoints defined on the currently active Text Edit Form [*cf.: command* `Debug > Clear Breakpoints`].

-<>-

## Notice (^) Menu

Context menu available on the rightmost column of the vertical ruler, where such "`Notice`" marks corresponding to syntax errors and warnings will be shown, as on the snapshot below.




[*see also: list of "*`Notice`*" types at the opening of sec.* Vertical Ruler, Context Menus]

– –

### *Command List*

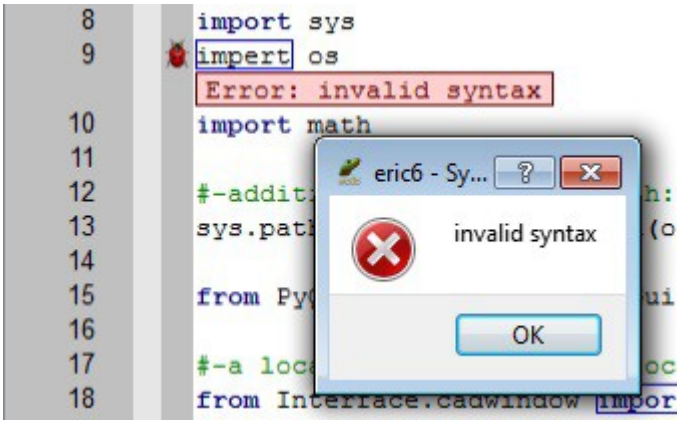
[*ref.:* `Bookmarks >]  Goto Syntax Error`
`Show Syntax Error Message  Clear Syntax Error`
[*ref.*[121]*:*  `Bookmarks >]  Next Warning [Message]  Previous Warning [Message]`
`Show Warning Message  Clear Warnings`
[*ref.* `Bookmarks >]  Next Uncovered Line  Previous Uncovered Line`
*ref.:* `Bookmarks >]  Next Task  Previous Task`
[*ref.:* `Bookmarks >]  Next Change  Previous Change`

– –

121<~> Same commands, just with a slightly different denomination [*a discrepancy we'd suggest to mend*].

## Notice (^) Show Syntax Error Message

Designed to show the message box associated to the currently pointed line of code, if affected by a syntax error [*cf. also:* `Notice (^) Show Warning Message`].




A click on the bug-mark will have the same result.

### *Remark*

A syntax error will be displayed at the loading and saving of a module, just one at a time, as the Python syntax checker will stop working after the very first error possibly encountered, and will be anyhow conveniently updated and refreshed automatically also while typing [*cf.: command* `Bookmarks > Goto Syntax Error`].

– –

## Notice (^) Clear Syntax Error

Designed to clear the syntax error possibly in display on the current module [*cf. also:* `Notice (^) Clear Warnings`]. Then next `File > Save` command[122] will anyhow refresh the Python syntax checking, and the consequent error display.

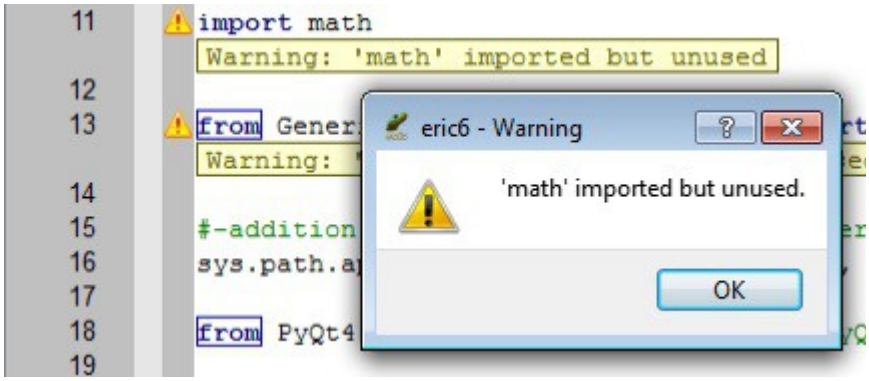
– –

122Hint: to re-enable a possibly disabled `File > Save` command you may just enter a dummy space on the Text Form.

## Notice (^) Show Warning Message

## Notice (^) Clear Warnings

Same function as the above corresponding "`Notice (^) Show Syntax Error Message`" and "`Notice (^) Clear Syntax Error`" context commands [*see*], in case of warnings.

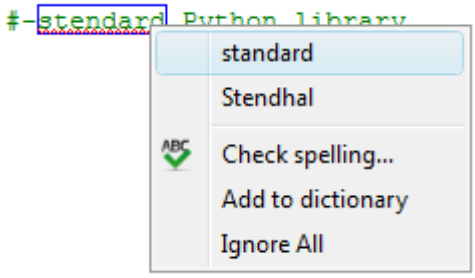



Note that the detection of a syntax error will take precedence over all warning conditions.

-<>-

# Check Spell Menu

Special pop-up context menu associated to the words intercepted by the Eric Spell-checker as possibly incorrect and, as such, marked with a squiggly red underline [*as with* stendard *hereafter*].




This feature requires the presence of the PyEnchant spell-checker toolkit [*see also:* `Settings > Show External Tools`].

## *Remark*

For a detailed description of this feature refer to the main menu command `Extras > Check Spelling...`[123], and also to the commands of the related sub-menu `Extras > Edit Dictionary` [*see*].

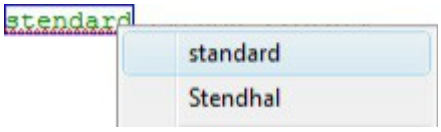
### *Command List*

`<Spell Aid>` `Check Spelling...` `Add to Dictionary` `Ignore All`

– –

## Spell (^) *<Spell Aid>*

List of spelling aids available to be possibly substituted to the pointed misspelled word, as on this example.

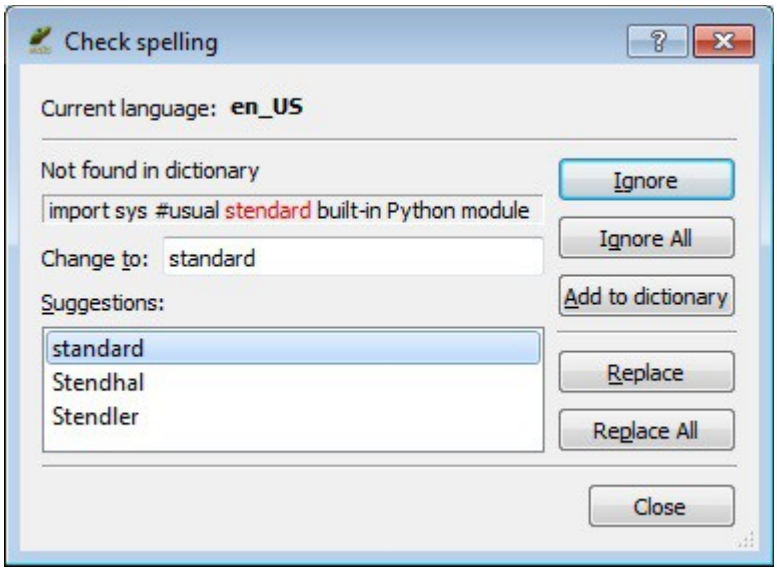



– –

123Functionally very similar but for the scope of the check, here focused on the single pointed word.

## Spell (^) **Check Spelling...**

Designed to show such a "`Check spelling`" control box:




where to manage the currently pointed spelling issue, as already described at the functionally similar main menu command: `Extras > Check Spelling...` [*see*].

– –

## Spell (^) **Add to Dictionary**

Command functionally identical to the corresponding button on the "`Check spelling`" control box [*see above*]. It refers to the currently pointed word squiggly underlined.

– –

## Spell (^) **Ignore All**

Command functionally identical to the corresponding button on the "`Check spelling`" control box [*see above*], designed to ignore all occurrences of the pointed spelling issue on the current text, and during the current edit session only[124]. It refers to the currently pointed word squiggly underlined.

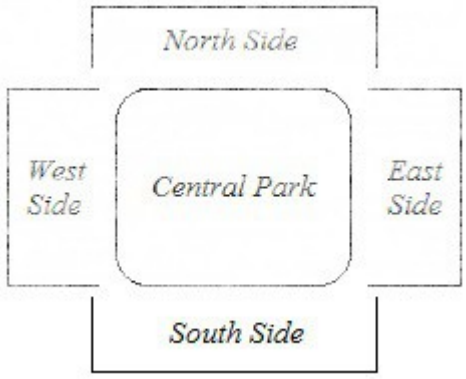
– = –

124That's why, in this case, there is no need for any general dictionary management feature, such as list and delete.

# {3.South} Eric Window – *South Side*

*S-PM 160400*

This {3.South} section is about the South Side area of the Eric application window, where are located the *Bottom Pane* and the *Information Bar* [*cf.:* Eric Window Map*, in:* {Map}*, at the end of this* Report], with the related context pop-up menus.




### *Context Menu List*

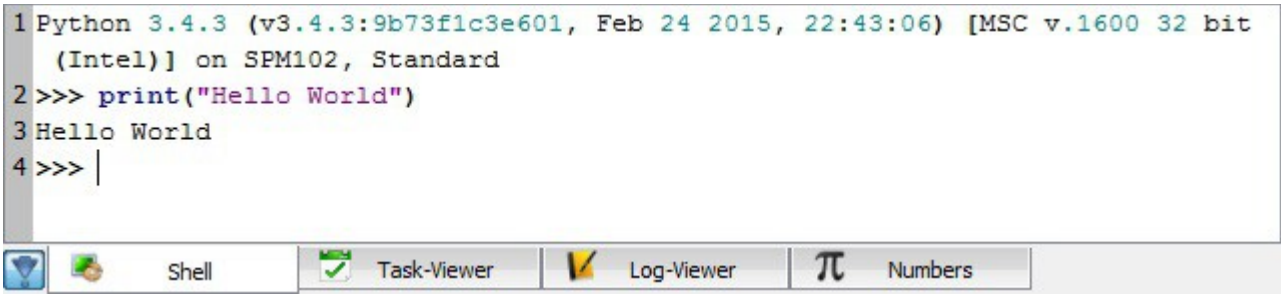
`Shell (^)` `Task (^)` `InfoBar (^)`

[*as with next sec.s:* Bt-Pane: Shell, Bt-Pane: Task-Viewer *and* Information Bar]

– –

## Auxiliary Bottom Pane

An optional tabbed form positioned at the bottom of the Central Pane, hosting the following set of tabs, ordered from left to right: `Shell`, `Task-Viewer`, `Log-Viewer`, `Numbers`




Tabs described in detail on the next sections.

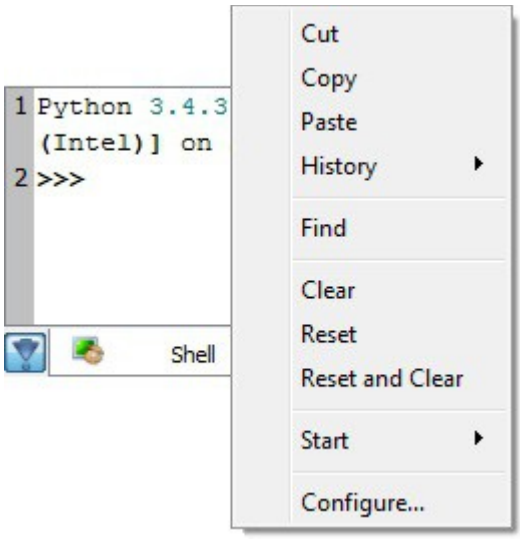
### *Remark*

The presence of this Bottom Pane[125] is optional—that is, not automatic—in the sense that it can be controlled with the main menu command `Window > Bottom Sidebar` [*see*] and, when in display, with a toggle-control to possibly activate an automatic collapsing mechanism.


– –

## *Bt-Pane*: **Shell**

"`Shell`" tab form, where to show the standard Python Command Shell, synchronized with the central Source Text Form and enriched with a pop-up context menu.




### *Command List*

[*ref.:* `Edit >`] `Cut` `Copy` `Paste` `History`

[*ref.:* `Edit > Search > Search...`] `Find` `Clear` `Reset` `Reset and Clear` `Start` `Configure...`

– –

125"Bt-Pane" hereafter, for short.

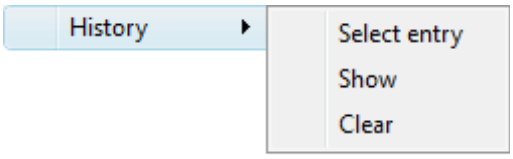
### *Remark*

Description of context commands identical or equivalent to those that can be found in the main Menu Bar [*see in:* {1.North}] will be there referred to, and not hereafter repeated. Also possible minor differences, due to the context condition, will be not treated, and left to the active user's interpretation.

– –

## Shell (^) **History**

Designed to show a sub-menu aimed at making use of the collection of statements last entered into the Command Shell.



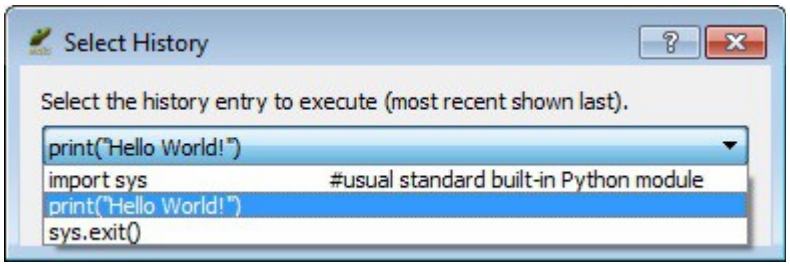

### *Sub-Command List*

`Select Entry` `Show` `Clear`

– –

## Shell (^) History > **Select Entry**

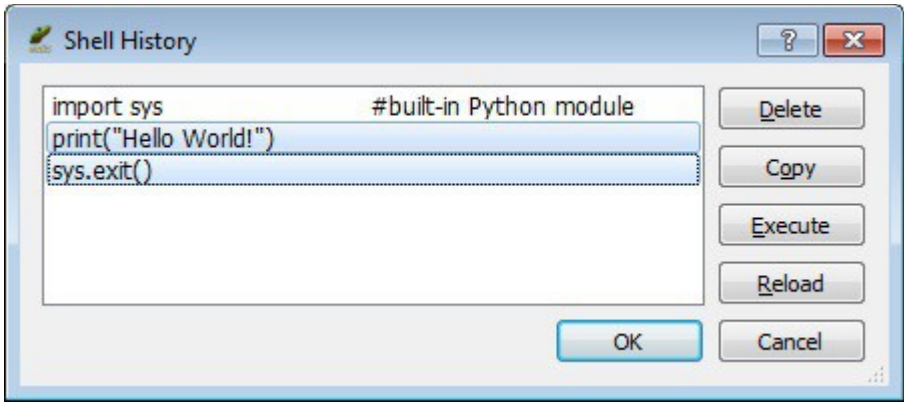
Designed to show a "`Select History`" control form with a drop-down list of the current history of statements entered into the Command Shell, from which to possibly pick one to be entered again.




– –

## Shell (^) History > **Show**

Designed to show a "`Shell History`" control box aimed at managing and using the currently recorded history of the statements entered into the Command Shell.




Related control buttons here assumed as self-explanatory; just note that:

- More than one line can be selected together, and operated upon.
- "`Copy`" button is assumed to copy selected lines into the currently active Text Edit Form; button therefore disabled in case of no line here selected.

– –

## Shell (^) History > **Clear**

Designed to erase the current history list of statements entered into the Command Shell.

– –

## Shell (^) **Clear**

Designed to clear the Python Command Shell an bring it back to the initial condition. Currently defined Python objects will remain unaffected.

– –

## Shell (^) **Reset**

Designed to reset the Python Command Shell execution status. Current text contents of the shell will remain unaffected.

– –

## Shell (^) **Reset and Clear**

Designed to combine both effects of the above `Clear` and `Reset` commands [*see*].

– –

## Shell (^) **Start**

Designed to show a sub-menu where to set which one of the listed languages is to be used into the Command Shell, languages possibly available beyond the very Python version in use as a prerequisite.


Selected language will then remain enabled at the next Eric re-start.

### *Remark*

<!> Note, in particular, that this command implies that Eric (6) is a true Python 2 and 3 compatible IDE, as already seen with the "`Progr. Language`" choice on the command `Project > Properties...` control box [*see*].

– –

## Shell (^) **Configure...**

Designed to show the same `Settings > Preferences...` window [*see*], just conveniently opened on the "`Configure Shell`" control box.

-<>-

## *Bt-Pane:* Task-Viewer

"`Task-Viewer`" tab form, where to show and manage the so called "*Tasks*", that is the user annotations about project development actions assumed as to be carried on in future times.

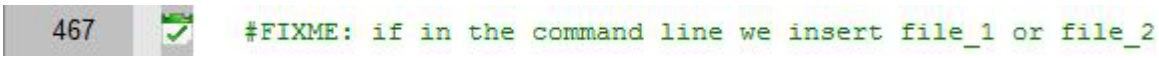
– –

There are two different types of such Tasks:

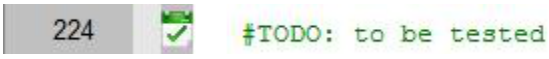
*Extracted T*ask[126] Design annotation added to the very source code text, in form of special Python comments such as: `#FIXME: …` and `#TODO: …`, as hereafter described in the following Remark section [*see also:* `Bookmarks > Next / Previous Task`].

### *Remark*

Precise format of the prefix tag for these special Extracted Task comment lines, such as the hereafter shown "`FIXME:`"




and "`TODO:`"




should be exactly as stated on: `Settings > Preferences... – Tasks, Tasks Markers` where, beside these two most popular ones, you'll see also a "`WARNING:`" and "`NOTE:`" marker [*see*].

<。> Then note that a `File > Save` command is required to update the detection of all Tasks possibly last entered on the current software module, and also to update the consequent display of the related Extracted Task Markers on the Notice column [*see:* Vertical Ruler, Context Menus*, in:* {2.Central}].

– –

*Manual Task*[127] Design annotation stored apart from any specific source text, or even apart from any specific Project, created and managed with these context commands [*cf. hereafter:* `Task (^) New Task...`], and not appearing on the source text.

– –

126<。> So called as "extracted", that is deduced, from the source text.

Formerly: *CodeTag*, as tentatively proposed with Python Enhancement Proposal PEP 350 (in: 2005, status: Rejected).

127Formerly: *Design Task*, now on (ver. 6.1 up): "Manual Task", as created "manually" with `(^) New Task...` [*see*]. <~> Not so happy a denomination this new one, we'd say, being the "Extracted" Tasks exactly as much "manual" too.

Eric Tasks can also be classified as Local or Global, according to their scope:

*Local Tasks* (i.e.: *Local* to a Project) are the Manual Tasks so declared with a checked "`Project Task`" property [*see: "`Task Properties`" hereafter*] and, of course, also all Extracted Tasks which, being special Python comments, necessarily belong to a source module, possibly belonging to a Project; Global otherwise [*cf.: next point*]. Local (Project) Tasks are printed bold on the Task-Viewer list [*see*].

*Global Tasks* (i.e: not-Project) are the Manual Tasks with a unchecked "`Project Task`" property [*see: "`Task Properties`" hereafter*] and—curiously enough—also the Extracted Tasks of source modules when opened singularly, that is not as part of a Project[128]. Global (not-Project) Tasks are printed normal (not-bold) on the Task-Viewer list [*see*].

– –

## *Remark*

Note that this Project / Global scope property for Extracted Tasks is automatic and unchangeable, whereas for Manual Tasks it's custom, with a currently open Project, automatic and unchangeable otherwise.

<。> Local (Project) Tasks are recorded in such a `<Project>\_eric6project\*.e4t` file, Global (not-Project) in a `<User>\_eric6\eric6tasks.e4t` file.

– –

## *Much Ado*

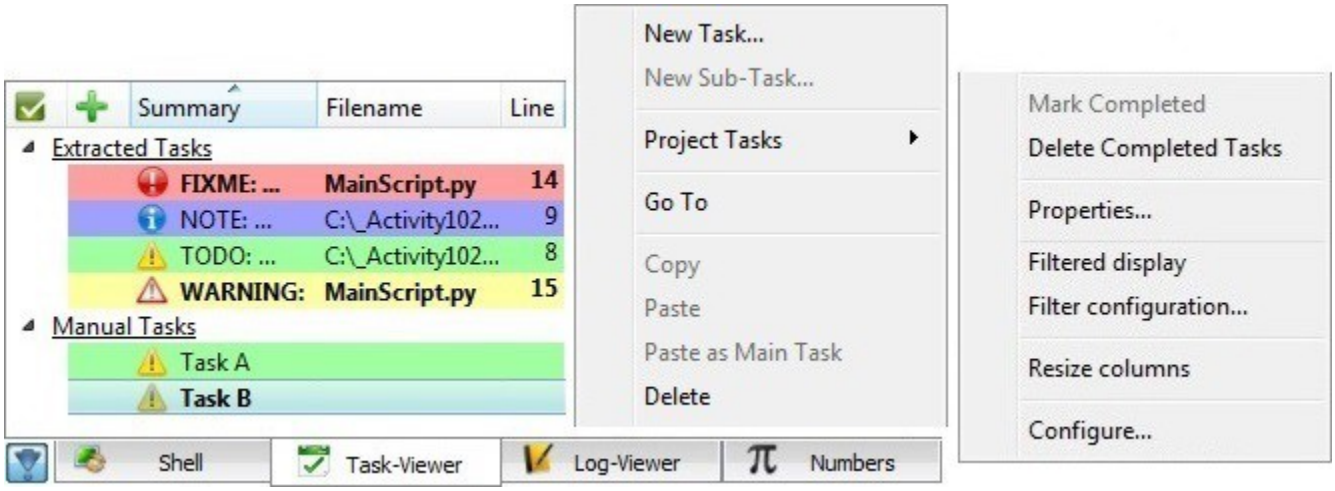
<~> Curious how rich and composite this Eric's *Task* feature is; for Tasks that, bottom line, are nothing more than plain sticky-notes. We'd love to know how it is perceived by the Eric's user community, whether as useful and adequate, or excessive.

-<>-

128<~> As you may easily verify looking at the related "`Project Task`" property check-box [*see in:* `Task (^) Properties...`]. There is a reason for that, as a self-standing module, as such, can be considered Global until not assigned to a single Projects.

## Task-Viewer Form, and Context Menu

A form for displaying a Tasks Table possibly listing both Extracted and Manual Tasks. Meaning of the table's columns is assumed as evident.




Just note that:

- The "`Line`" and "`Filename`" columns are significant only for Extracted Tasks, whereas such "✓" tick-sign (Task Completed) and "+" plus-sign (Task Priority) columns are significant only for the Manual ones [*see:* `Task (^) Properties...`, *"*`Project Task`*" check box*].
- Local (Project) Tasks' line text is here printed bold, Global Tasks normal.

### *Command List*[129]

```
    New Task...[130]        Next Sub-Task...          Project Tasks    Go To
[ref.: Edit >] Copy         Paste
    Paste as Main Task
[ref.: Edit >] Delete
    Mark Completed          Delete Completed Tasks    Properties...
    Filtered Display        Filter Configuration...   Resize Columns   Configure...
```

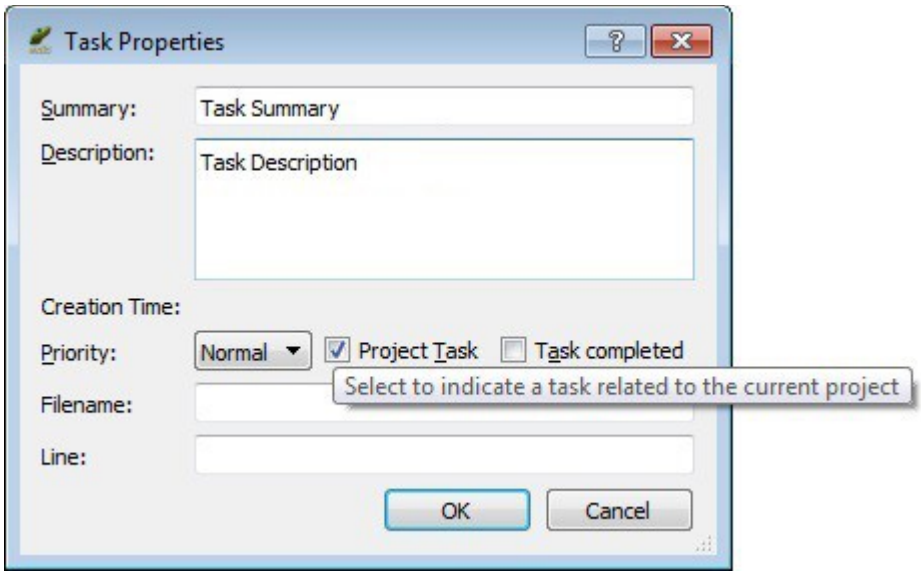
– –

129<。> Note that this command list is reduced when no Task is selected.

130<~> "Manual Task" intended only, not the Extracted. Whereas with such other command as `Bookmarks > Next / Previous Task` [*see*] is intended the Extracted only. Confusion & ambiguity that we suggest should be fixed.

## Task (^) **New Task...**[131]

Designed to show a "`Task Properties`" dialog box




where to define a new *Manual Task*, that is a Task of different format from such `#TODO:` and `#FIXME:` *Extracted Task*, as described at opening of the above ***Bt-Pane:*** Task-Viewer section [*see*]. Such Manual Tasks are intended as annotations for general developments not necessarily tied to any given source module, nor Project. <!> That's why this context command is available and enabled also with of an empty Text Edit form.

This "`Task Properties`" box is the same for the context command `Task (^) Properties...` [*see*]; with, in particular:

`Project Task` Check-box enabled only in case of "`New Task`" created with a Project opened, so to possibly decide whether to associate it to the current Local Project, or not. Otherwise, in all other cases, automatically pre-set and disabled; indeed, note that:
- Manual Task cannot be assigned to a Project if created with no opened Project;
- Extracted Tasks are necessarily associated to the related source text, and also to the related opened Project, if any.

`Filename` and `Line`
Fields void and disabled, as not significant for this Manual Task case. Whereas significant, and read-only, for the Extracted Tasks.

– –

131<.> "Manual Task" here intended only, not the "Extracted".

### *Remark*

Manual Tasks created with this command are qualified as "Main", as they can hold sub-Tasks nested into them by means of `(^) Next Sub-Task...` command [*see next cmd, and also:* `(^) Paste as Main Task`].

– –

## Task (^) **Next Sub-Task...**

Designed to create a new Manual sub-Task nested into the currently selected one, either Main or a sub-Task [*cf.:* `(^) New Task...` *above*]; command disabled otherwise. Features:

- Command working as the parent `Task (^) New Task...` command [*see*];
- but for the "`Project Task`" box condition, inherited from the related parent Task.
- Sub-Tasks can be nested.

– –

## Task (^) **Project Tasks**

Designed to show a sub-menu of commands aimed at managing the generation of the Task-Viewer table for the current Project.


Disabled with no Project currently opened, in which case anyhow only the Global (i.e.: not-Project) Manual Tasks can be listed [*cf.:* `Project Task` *check-box, for the above* `(^) New Task...`].

– –

### *Sub-Command List*

`Regenerate Project Tasks` `Configure Scan Options`

– –

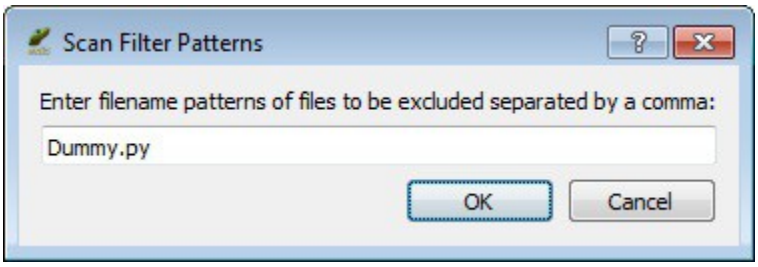
## Task (^) Project Tasks > **Regenerate Project Tasks**

Designed to refresh the Task-Viewer table according to the current condition, in particular after a new `Project Tasks > Configure Scan Options` setting [*see next cmd*].

– –

## Task (^) Project Tasks > **Configure Scan Options**

Designed to show a "`Scan Filter Patterns`" control box




where to enter filename patterns to be excluded from the Task-Viewer table, possibly comma separated and with usual "`*`" wildcards [*e.g.:* `ThirdParty\*,*\coverage\*`]. The changes possibly entered then require also a: `(^) Project Tasks > Regenerate Project Tasks` to take effect [*see previous cmd*].

Last entered Filter Patterns will be then in display when calling over this command.

– –

## Task (^) **Go To**

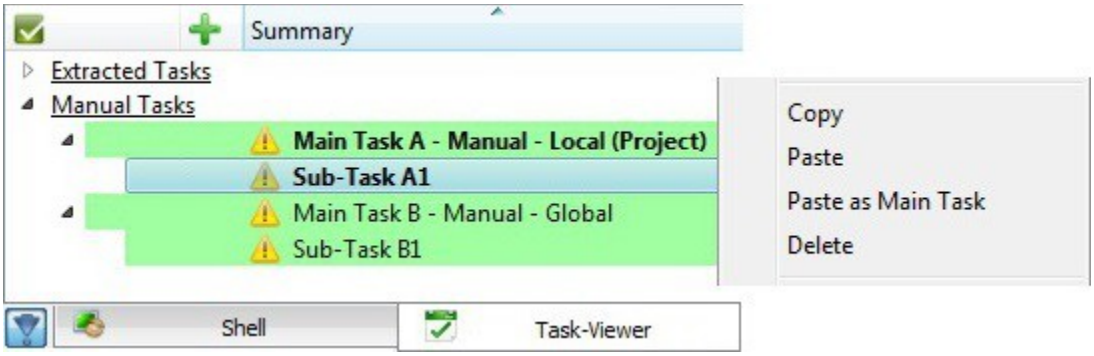
Designed to synchronize the Python module in display on the Text Edit Form with the selected Extracted Task source line; command disabled otherwise.

A double-click on the very Task-Viewer table line will have the same result.

– –

## Task (^) Paste as Main Task

Designed to paste a just `Copy`-ed Manual sub-Task as a Main Task, that is to duplicate it promoted at the top level [*cf.: above* `(^) New Task...` *cmd*].




Command enabled only for sub-Task.

### *Remark*

<。> Note that this particular sub-set of `Copy` – `Delete` context commands is meant and meaningful only for Manual Tasks, not for the Extracted ones.

– –

## Task (^) Mark Completed

Designed to mark the selected Manual Task as "Completed"[132].

Completion mark can be reset using the related `Task (^) Properties...` command [*see*].

– –

## Task (^) Delete Completed Tasks

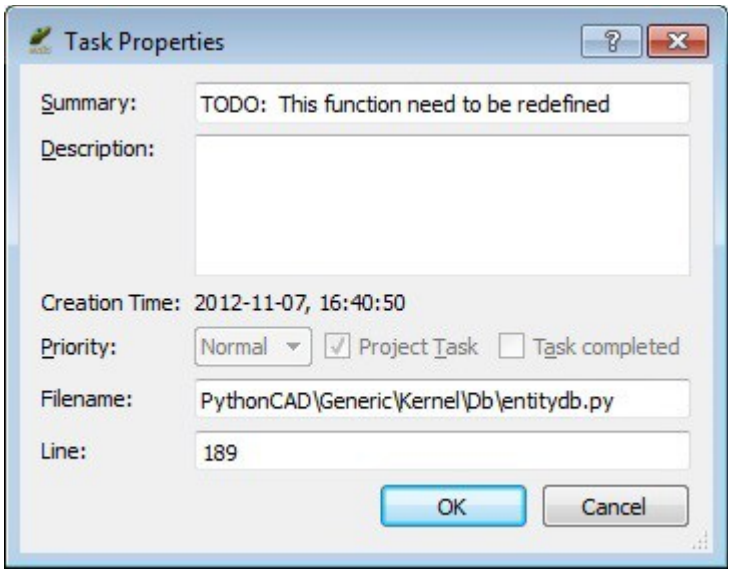
Designed to delete all Manual Tasks currently marked as Completed [*cf.: above* `Task (^) Mark Completed`].

– –

132Note that the Extracted Tasks do not need such "Completed" status management commands, as they are meant to be managed acting directly at the source code level.

## Task (^) **Properties...**

Designed to show a "`Task Properties`" dialog box




where to see and manage the properties of the currently selected Task.

Read-only, read-write and enabled field status will properly vary in case of Extracted or Manual Tasks [*cf.: definitions at the beginning of this* Bt-Pane: Task-Viewer *sec.*].

– –

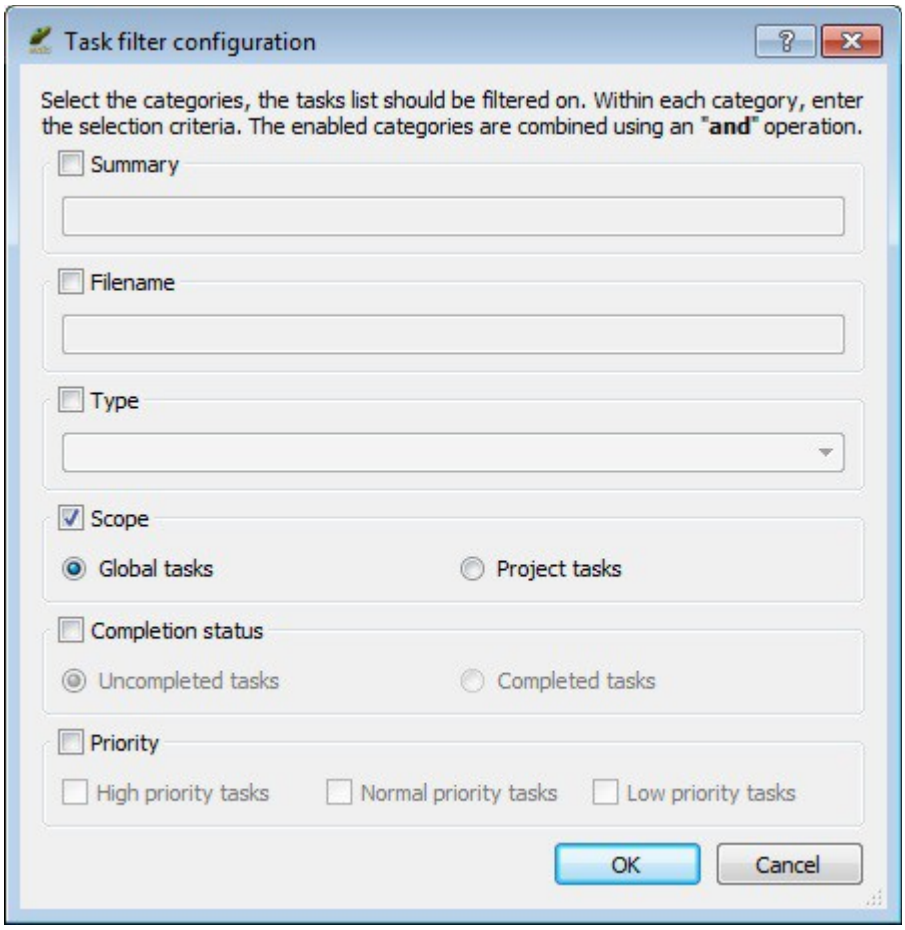
## Task (^) **Filtered Display**

Designed to enable / disable the `Task (^) Filter Configuration...` currently defined for the display of the Task-Viewer [*see: next cmd*].

– –

## Task (^) **Filter Configuration...**

Designed to show such "`Task filter configuration`" control box:




where to set the selection criteria for the Tasks to be listed in case of `(^) Filtered Display` enabled [*see: previous cmd*].

This is a control box assumed as clear enough not to require any further explanation but, perhaps, a comment to the term "`Global`", as opposed to "`Project`", here typically referred to the Manual Tasks not assigned to any specific Project [*cf.: definitions at the beginning of this* Bt-Pane: Task-Viewer *sec.*].

– –

## Task (^) **Resize Columns**

Designed to reset the automatic width of the Task-Viewer table's columns, possibly hand-modified.

– –

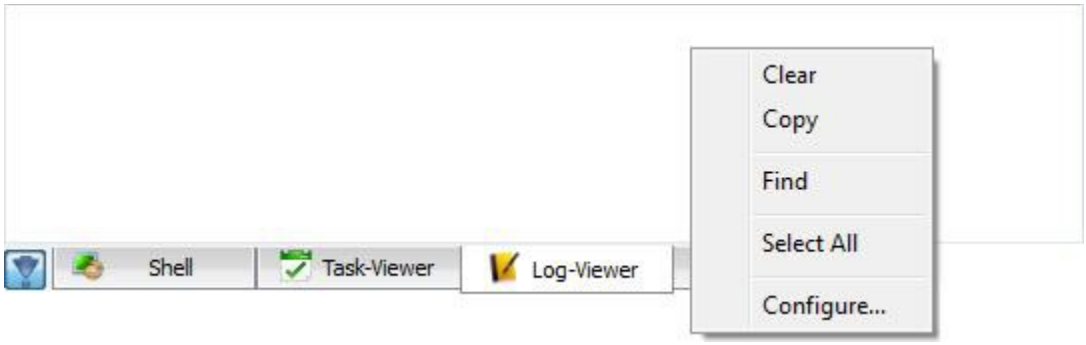
## Task (^) **Configure...**

Designed to show the same `Settings > Preferences...` window [*see*], just conveniently opened on the "`Configure Tasks`" control box.

-<>-

## *Bt-Pane:* Log-Viewer

"`Log-Viewer`" tab form, where Eric is assumed to print and show its own "Standard Output" and "Standard Error" messages[133], respectively in black and red colored text, so to tell the difference.




Related context menu is of immediate comprehension, so not to require any particular description.

– –

### *Remark*

<。> This Viewer is for messages specifically generated by Eric and by tools used directly by Eric[134].

Whereas, for instance, such a plain Python fragment as:

```
import sys
sys.stdout.write("Something")
```

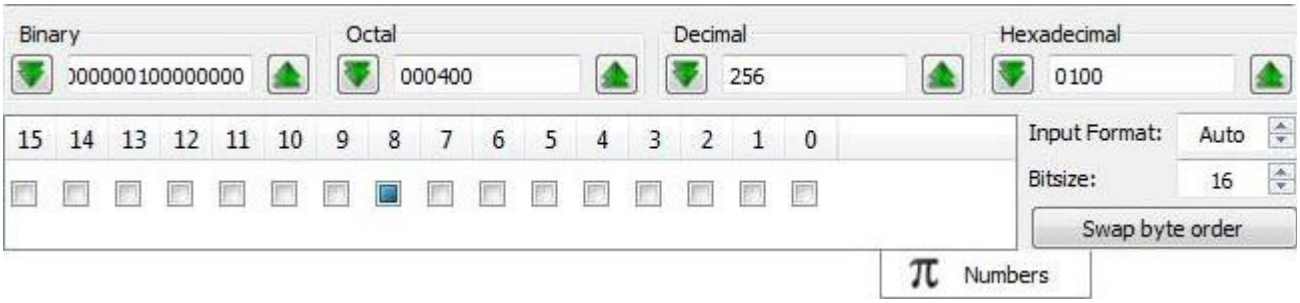
will print its `"Something"` string onto the usual "`Shell`" tab form [*see above:* Bt-Pane: Shell *sec.*], that is not in here.

-<>-

133That is: "`stdout`" and "`stderr`" devices, saying it with usual Unix terms.
134Such as, we've been told: PyLupdate and iRelease, when dealing with translation strings.

## *Bt-Pane:* **Numbers**

"`Numbers`" tab form, where to show a tool box aimed at representing and converting numbers into various formats, as it may come handy to programmers. Numbers that can also be conveniently interchanged with the source text in display on the current Text Edit Form.




### *Specifications:*

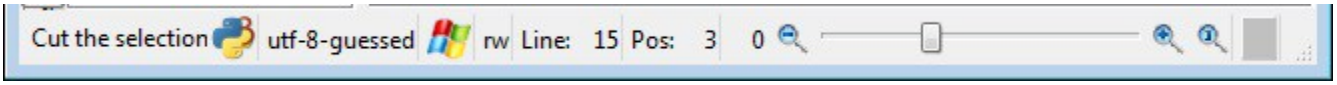
Buttons provided for copying selected numbers from / to the Text Edit Form;

`Swap byte order` Button provided and handy for a "`Bitsize`" value greater than 8 [*verify*].

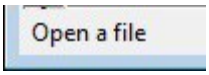
-<>-

# Information Bar

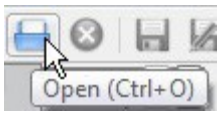
Information or Status Bar, aimed at displaying some information related with the text form currently active on the central pane [*see:* Eric Window Map, *in:* {Map}].




### *Info List*




Eric information message, typically to describe the action immediately available under the mouse pointer. Here an example:



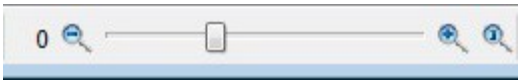

Syntax highlighter language [*cf.:* `Text Form (^) Languages`, *in:* {2.Central}].
<。> A click on this icon is equivalent to the cited context command [*verify*].


Unicode character encoding code [*cf.:* `Text Form (^) Encodings`, *in:* {2.Central}].
<。> A click on this icon is equivalent to the cited context command [*verify*].

End-Of-Line Type [*cf.:* `Text Form (^) End-Of-Line Type`, *in:* {2.Central}].
<。> A click on this icon is equivalent to the cited context command [*verify*].


Read-Write ("`rw`") or Read-Only ("`ro`"), as is the current text file permission.


Current line & character insertion point position.




Text zoom manager [*cf.:* `View > Zoom`, *in:* {1.North}].

VCS Status Monitor LED [*see:* `Project > Version Control` *main cmd*].
Color coded as shown by the `Shift+F1` help box [*see*].

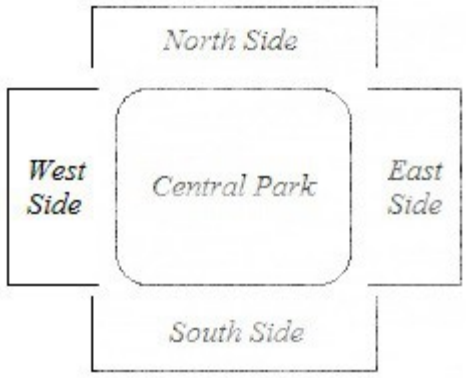
– = –

# {4.West} Eric Window – *West Side*

*S-PM 160400*

## Auxiliary Left Pane

Here we are at the "West Side" area of the Eric Application Window [*cf.:* Eric Window Map*, in:* {Map}*, at the end of this* Report].




An optional pane positioned at left on the Eric window[135], hosting a tabbed form comprising the following vertical tabs, ordered from top to bottom and described in detail on subsequent sections:

```
Project-Viewer, Multiproject-Viewer, Template-Viewer, File-Browser, Symbols
```


135"L-Pane" hereafter, for short.

Optional in the sense that its presence can be controlled with the main menu command `Window > Left Sidebar` and selected in view with the `Window > Edit / Debug Profile` as configured on the "`Configure View Profiles`" control box of command `Settings > View Profiles...` [*see*].

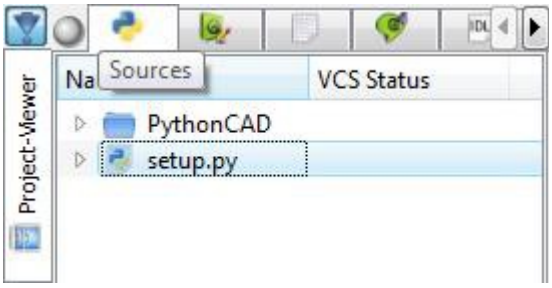

Plus, when in display, also with the usual toggle-control button for the automatic collapsing mechanism.

-<>-

## *L-Pane*: **Project-Viewer**

A tabbed form, hosting the following set of tabs ordered from left to right, described in detail on subsequent sections:

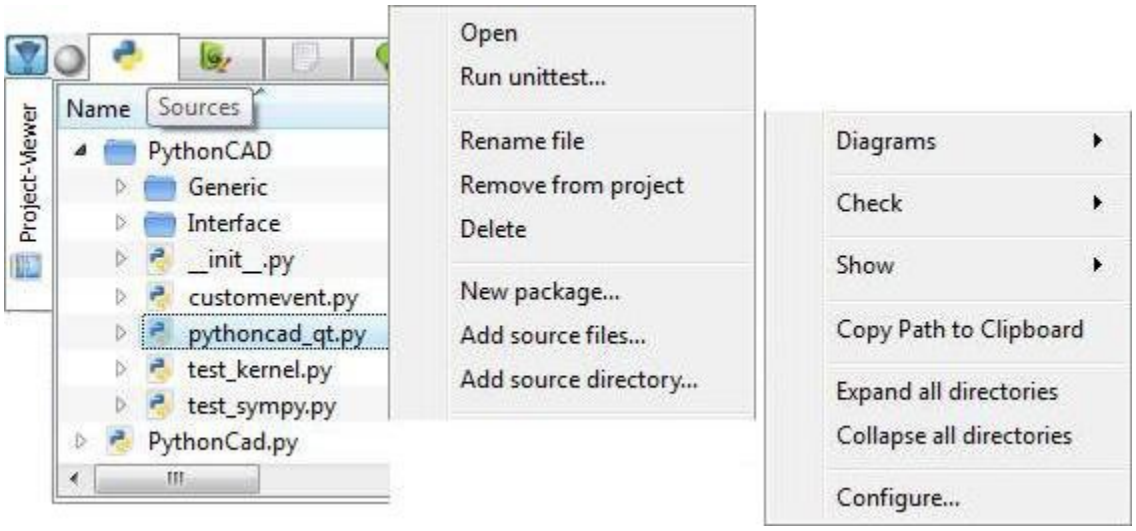
`Sources`, `Forms`, `Resources`, `Translations`, `Interfaces (IDL)`, `Others`




Note that this `Project-Viewer` form becomes enabled only when an actual Eric Project happens to be `Open` [*see: main menu* `Project > …`]

– –

## *Project-Viewer:* **Sources**



### *Command List*

[*ref.:* `File >`] Open
[*ref.:* `Unittest >`] Run unittest...
Rename File Remove from Project Delete
New Package... Add Source Files... Add Source Directory...
[*ref.:* `Project >`] Diagrams[136] Check Show
[*ref.:* `Tab (^)`, *in:* {2.Central}] Copy Path to Clipboard
Expand / Collapse All Directories Configure...

– –

## Sources (^) **Rename File**

Designed to rename the selected file. Command disabled with Version Controlled Projects [*cf.:* `(^) Remove from Project` *below*], as an action to be done necessarily through the selected Version Control System [*see: section* Project Command Menu, *property* `Version Control System`; *in:* {1.North}].

<!> Command to be used anyhow with due care as you are dealing with Projects, where file names count, and synchronization of the changes on names is not automatic.

– –

## Sources (^) **Remove from Project**

Designed to simply exclude the selected file from the current Eric Project and, therefore, from the display on the current `Sources` list. But anyhow with the very file, and its possible role as a Python source module, that will remain unaltered[137]. The assignment to an Eric Project can anyhow be reset by means of command `Sources (^) Add Source Files...` [*see*].

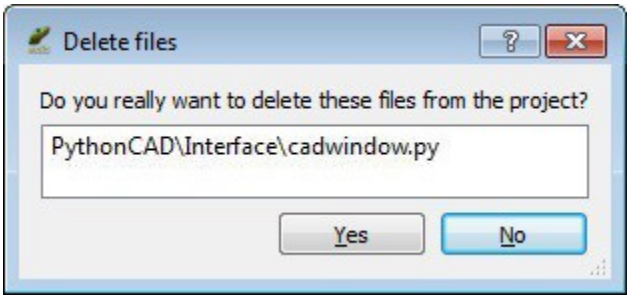
Command disabled with Version Controlled Projects [*cf.:* `(^) Rename File` *above*], as an action to be done necessarily through the selected Version Control System [*see: section* Project Command Menu, *property* `Version Control System`; *in:* {1.North}].

– –

136<~> For some reason this context sub-menu is richer than the referred main menu correspondent, as it comprises these other sub-commands: `Class / Package / Imports Diagrams...` [*see*]. Anyhow we still consider this as a specialized topic, out of this Report's scope, and about which here we have nothing else to add to what already said.
137<。> That's coherent with the fact that Project is a specific Eric concept, with no precise correspondence in Python.

## Sources (^) **Delete**

Designed to actually delete the files currently selected on the `Project-Viewer` form.

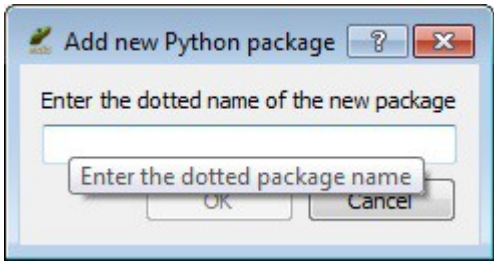



<!> Beware: it's a *real deletion* from the very computer file system, and not simply from the shed of the Eric Project, as this "`Delete files`"-box caption—specifying: "`from the project`"—seems to suggest. In case you'd simply want to have the selected files not considered as part of the Project and in display on the `Sources` list, consider the above context command `(^) Remove from Project` [*see*].

– –

## Sources (^) **New Package...**

Designed to show an "`Add new Python package`" control box, so to create a new *Package* as intended by Python [*and as hereafter defined*].




A Python Package, that is: a directory with that name, within the current Eric Project, with a "`__init__.py`" Module inside, typically aimed at hosting a collection of source files and, possibly, other sub-Packages.

### *Specifications:*

"`dotted name`" Same syntax as usual with the `import` Python statement [*see*]; e.g.: `myLib.subLib`

– –

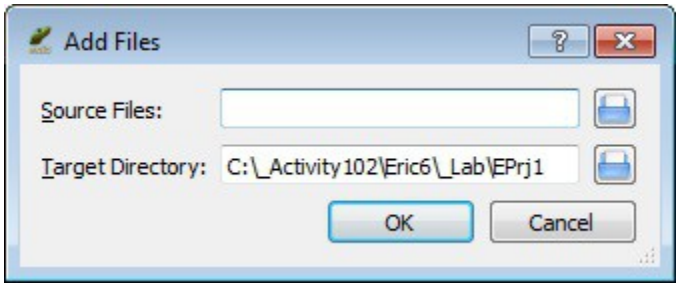
### *Remark*

Then, to possibly fill such Package with existing source files, next context commands `(^) Add Source Files...` and `(^) Add Source Directory...` come just handy [*see*].

– –

## Sources (^) **Add Source Files...**

Designed to show an "`Add Files`" control box so to possibly add some other existing files to the current Project [*cf.: command* `(^) Add Source Directory...`, *and also:* `(^) New Package...`].




In case of distinct source and destination directories, the named files are actually *copied* into the `Target Directory`, with the original files left unaltered.

### *Remark*

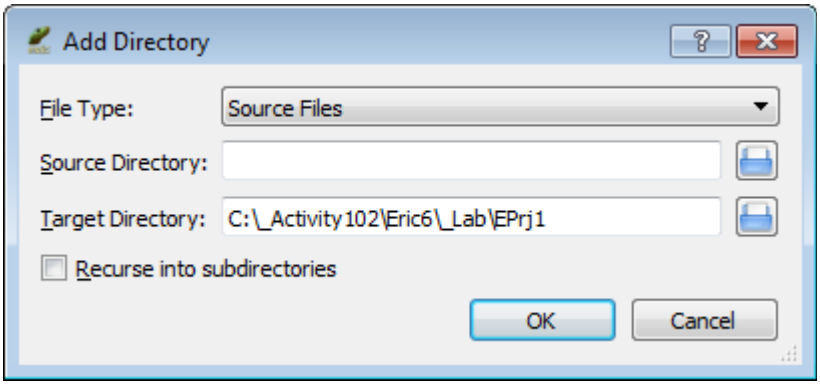
A pair of practical suggestions:

- Not only source files can be here searched and added to a Project, but *ANY* type of file; just selecting the "`All Files (*)`" extension type when searching the `Source Files` directory.
- Entering full qualified paths directly into the "`Source Files`" field, instead of making use of the search button, more than one file at a time can be added.

– –

## Sources (^) **Add Source Directory...**

Designed to show a "`Add Directory`" box so to possibly add—that is: *copy*—to the current Project all source files contained into a given source directory. All original items will remain unaltered [*cf. command:* `(^) Add Source Files...`, *and also:* `(^) New Package...`].




*Specifications:*

`File Type – Source Files`

As they are defined with main command `Project > Filetype Associations...` [*see*].

– –

## Sources (^) **Expand All Directories**

## Sources (^) **Collapse All Directories**

Designed to expand / collapse all directories in display on the `Project-Viewer`.

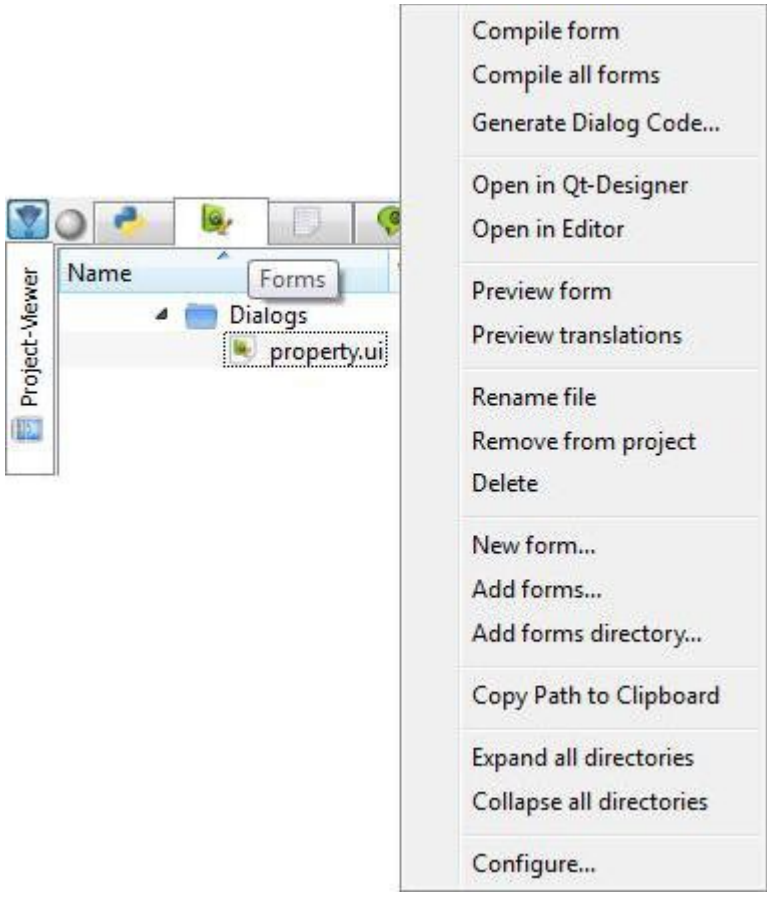
– –

## Sources (^) **Configure...**

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure project viewer settings`" control box.

-<>-

## *Project-Viewer:* **Forms**




### *Command List*

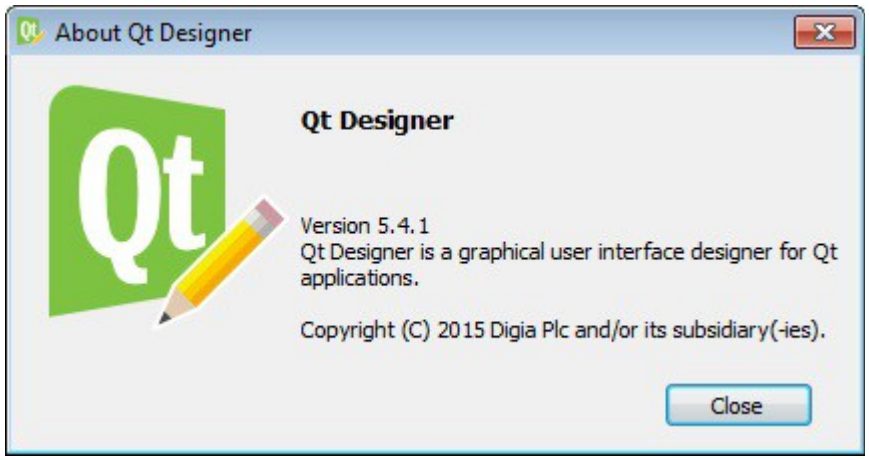
```
     Compile Form  Compile All Forms  Generate Dialog Code...
     Open in Qt-Designer    Open in Editor   Preview Form     Preview Translations
[ref.: Sources (^)]  Rename File        Remove from Project    Delete
[ref.: Sources (^)]  New / Add Forms    Add Forms Directory...
[ref.: Tab (^), in: {2.Central}]   Copy Path to Clipboard
[ref.: Sources (^)]  Expand / Collapse All Directories         Configure...
```

### *Remark*

A self-standing subject of specific relevance with a highly specialized set of context commands, here not treated being assumed off scope for this Report, as better clarified hereafter [*see*].

– –

A graphical designing environment for Qt forms—that is: Qt "`*.ui`" User Interface type files—centered onto the "Qt Designer" tool,




as with the main menu command `Extras > Builtin Tools > Qt-Designer...` [*see*].

Actually a central designing tool equipped with a contour of other tools, as shown in this context-menu [*see*], and whose behavior can be variously enriched and customized with other optional Eric plug-ins [*see:* Plugins Command Menu, *in:* {1.North}].

– –

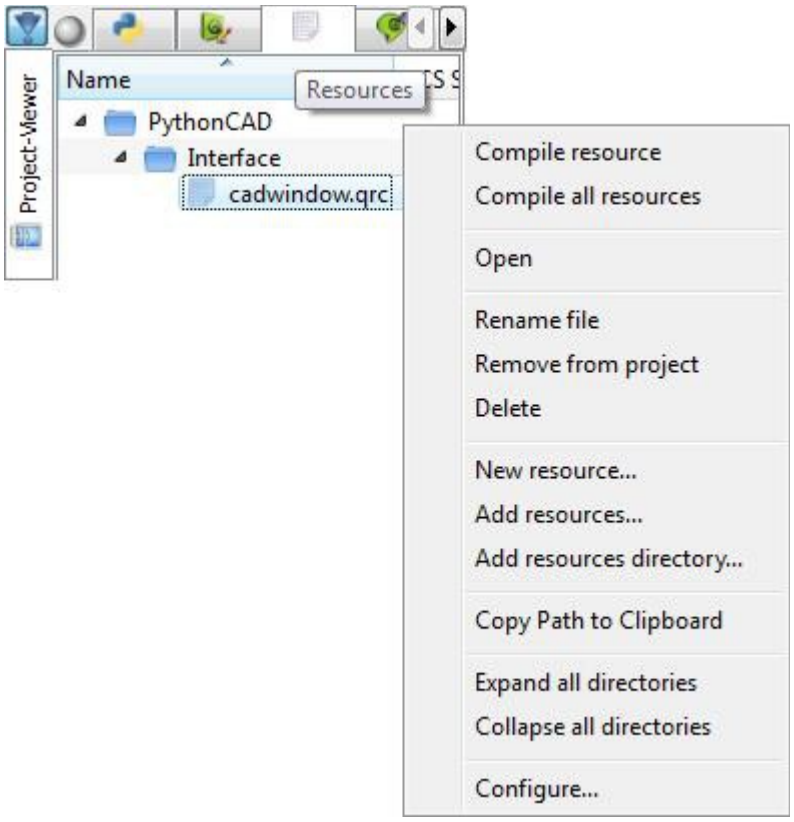
Certainly a relevant and engaging subject, that would deserve a dedicated Eric booklet of its own [*for a general reference see:* Reference Book List, *in:* Appendix], but here just hinted at, as assumed off the scope of this Report [*see:* Scope of this Report, *in:* {0.Lead}].

-<>-

## *Project-Viewer:* **Resources**

A designing environment for Qt Resource Collection "`*.qrc`" type files, centered onto the "Qt Resource System" tool, defined in the Qt documentation as: "a platform-independent mechanism for storing binary files in the application's executable. This is useful if your application always needs a certain set of files (icons, translation files, etc.) and you don't want to run the risk of losing the files."

Plus equipped with a contour of other tools as shown in the context-menu [*see*], whose behavior can be variously enriched and customized with optional Eric plug-ins [*see:* Plugins Command Menu, *in:* {1.North}].




### *Command List*

```
     Compile Resource          Compile All Resources   Open
[ref.: Sources (^)] Rename File  Remove from Project     Delete
[ref.: Sources (^)] New / Add Resources                  Add Resources Directory...
[ref.: Tab (^), in: {2.Central}]      Copy Path to Clipboard
[ref.: Sources (^)] Expand / Collapse All Directories    Configure...
```

– –

### *Remark*

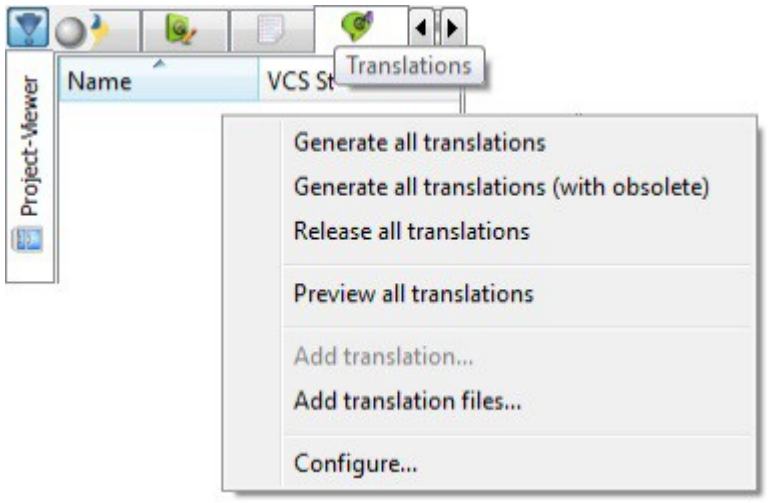
A self-standing subject of specific relevance with an highly specialized set of context commands, here not treated being assumed off scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

– –

## *Project-Viewer:* **Translations**

This Eric feature has mainly to do with Qt Linguist[138], that is a tool for adding translations to Qt applications, also available as an Eric builtin tool [*see:* `Extras > Builtin Tools > Qt-Linguist...` *and also:* `Project > Add Translation...`].

Plus equipped with a contour of other tools as shown in the context-menu [*see*], whose behavior can be variously enriched and customized with optional Eric plug-ins [*see:* Plugins Command Menu, *in:* {1.North}].




No further detailed description of this feature will be here offered, as assumed off-scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

### *Command List*

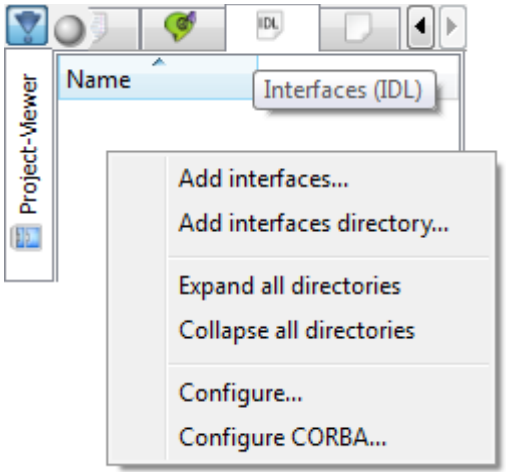
| | | |
|---|---|---|
| | `Generate All Translations` | `Generate All Translations (with Obsolete)` |
| | `Release All Translations` | `Preview All Translations` |
| [*ref.:* `Sources (^)`] | `Add Translation...` | `Add Translations Files...` |
| [*ref.:* `Sources (^)`] | `Configure...` | |

– –

138Plus, we've been told: PyLupdate and iRelease.

## *Project-Viewer:* Interfaces (IDL)

This Eric feature has to do with the *Interface Definition Language*, IDL for short, which is a specification language used to describe a software component's interface in a language-neutral way, therefore enabling communication between software components that do not share a coding language.




The here named Common Object Request Broker Architecture (CORBA) is a standard set of rules defined by the Object Management Group (OMG) to enable software components, written in multiple computer languages and running on multiple computers, to work together.

No further detailed description of this feature will be here offered, as assumed off-scope for this Report [*see:* Scope of this Report, *in:* {0.Lead}].

### *Command List*

[*ref.:* `Sources (^)`] `Add Interfaces...` `Add Interfaces Directory...`
[*ref.:* `Sources (^)`] `Expand / Collapse All Directories` `Configure...` `Configure CORBA...`

– –

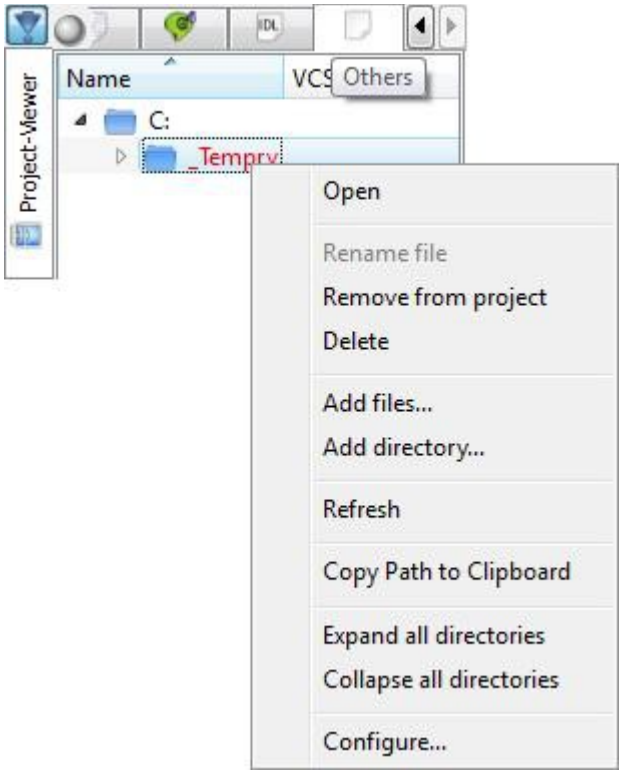
## Interfaces (IDL) (^) Configure CORBA...

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure CORBA support`" control box.

-<>-

## *Project-Viewer:* **Others**

Designed to assign to an Eric Project "any" other file, or directory, not belonging to the categories so far considered.




### *Remark*

<!> Rather a remarkable feature, that extends the concept and role of the Eric Project beyond that of a sheer container of Python source files, as it could be initially perceived. Note also that such Project is a specific Eric concept, with no precise correspondence in Python terms.

### *Command List*

```
[ref.: File >]      Open
[ref.: Sources (^)] Rename File    Remove from Project    Delete
[ref.: Sources (^)] Add Files...   Add Directory...
     Refresh
[ref.: Tab (^)]     Copy Path to Clipboard
[ref.: Sources (^)] Expand / Collapse All Directories    Configure...
```

– –

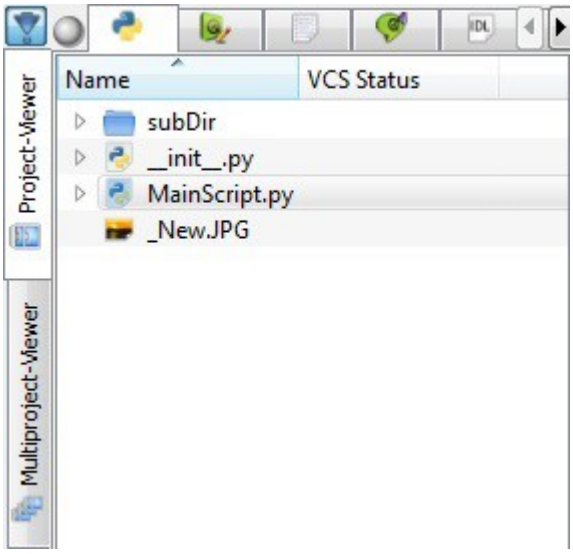
### Others (^) **Refresh**

Designed to update the contents of this `Project-Viewer: Others` form, as possibly dynamically modified by external actions.

– –

## *L-Pane*: **Multiproject-Viewer**

Just an extension of the similar `Project-Viewer` form [*see above*], whose reference is to be considered valid here too.

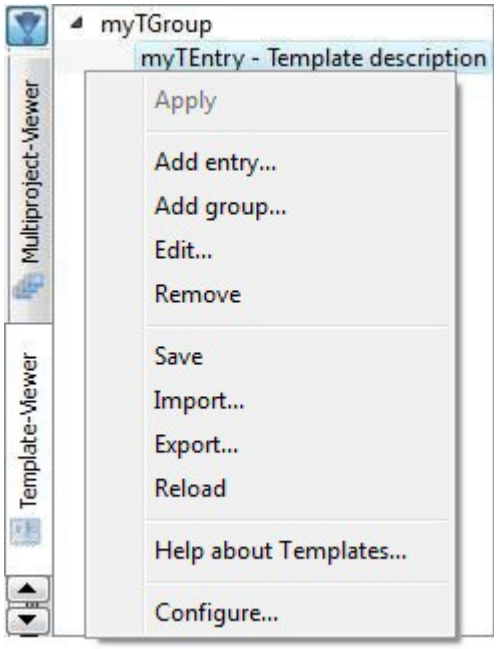



Same way the `Multiproject` main menu item has been here considered, and treated, that is as a straightforward extension of the `Project` menu item [*see*].

-<>-

## *L-Pane*: **Template-Viewer**

A feature aimed at managing and using the Eric *Templates*, that is a collection of custom source text fragments, possibly containing parameters, ready to be pasted into any source module[139] [*cf.:* `Extras > Macros`, *a similar feature, less powerful*].




### *Command List*

`Apply` `Add Entry...` `Add Group...` `Edit...` `Remove`
`Save` `Import...` `Export...` `Reload` `Help about Templates...`
`Configure...`

– –

### Template (^) **Apply**

Designed to insert the selected Template into the currently active source text, at the current insertion point. A double-click on the Template will get the same effect. Disabled if no edit text open.

In case of presence of Template variables [*see:* `Template (^) Add Entry...`, `Help` *button*], they will be asked and resolved at this insertion time.

139<。> Consider giving first a look at the context command `(^) Add Group...` as a good starting point in case this interesting feature is new to you.

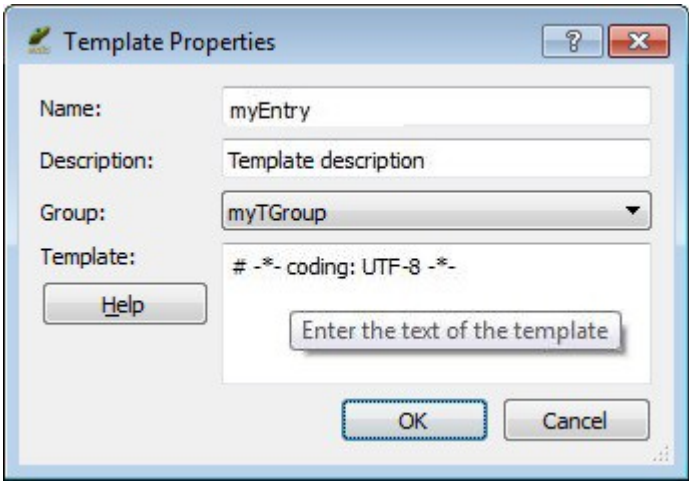
### *Remark*

For an even more direct insertion method see the Remark on the next `Template (^) Add Group...` command.

– –

## Template (^) **Add Entry...**

Designed to show a "`Template Properties`" control box where to create a new Template to be added into the currently pointed Group [*see:* `Template (^) Add Group...`].



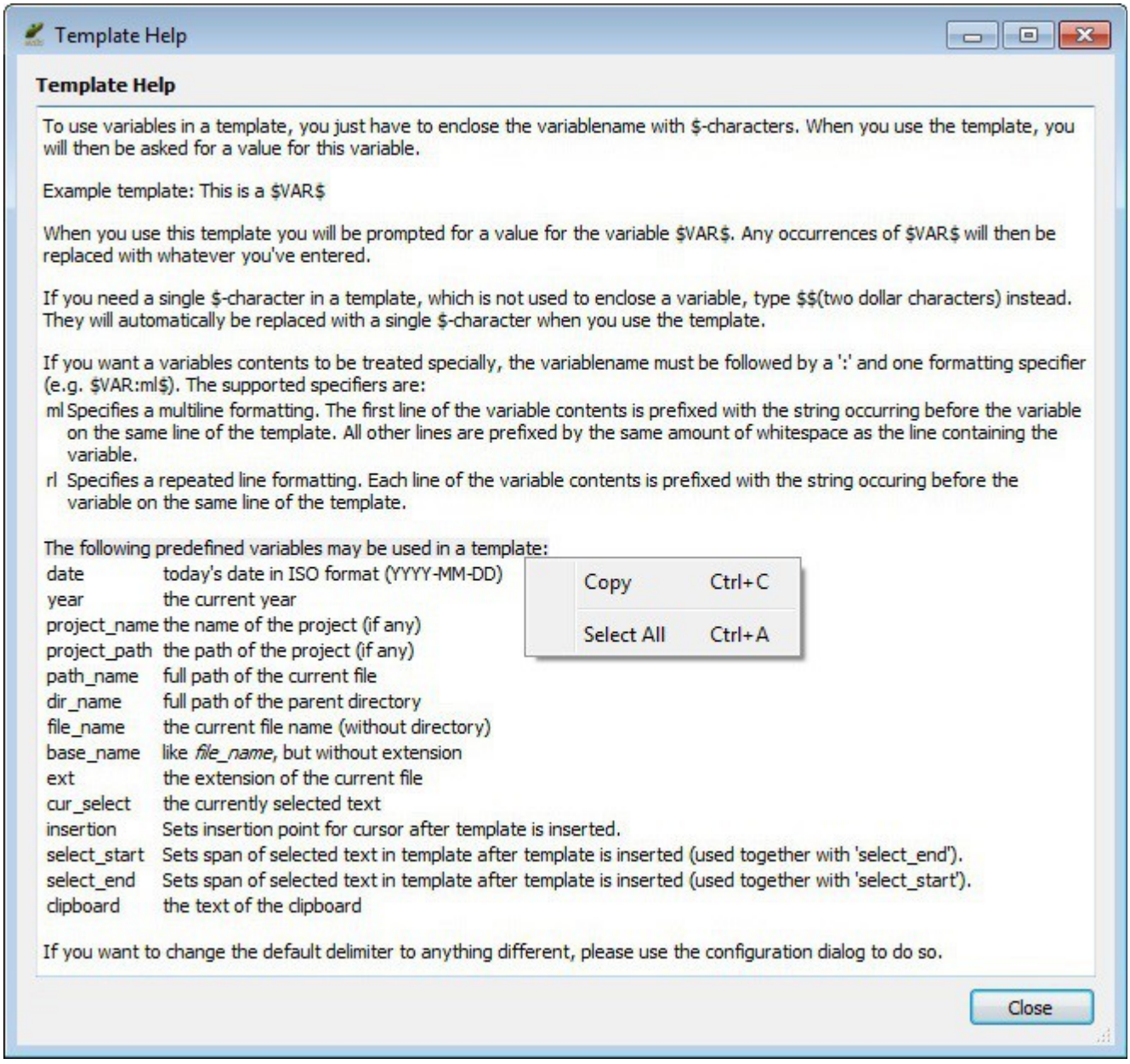

Desired source text can be entered, that is typed and edited or pasted, right away into the `Template` field, with the aid of a handy pop-up context menu [*verify*].

– –

### *Remark*

The `Help` button is here to display the following essential Eric "`Template Help`" instruction box, useful if you want to know how to create Templates possibly enriched with variables to be then resolved at insertion time.

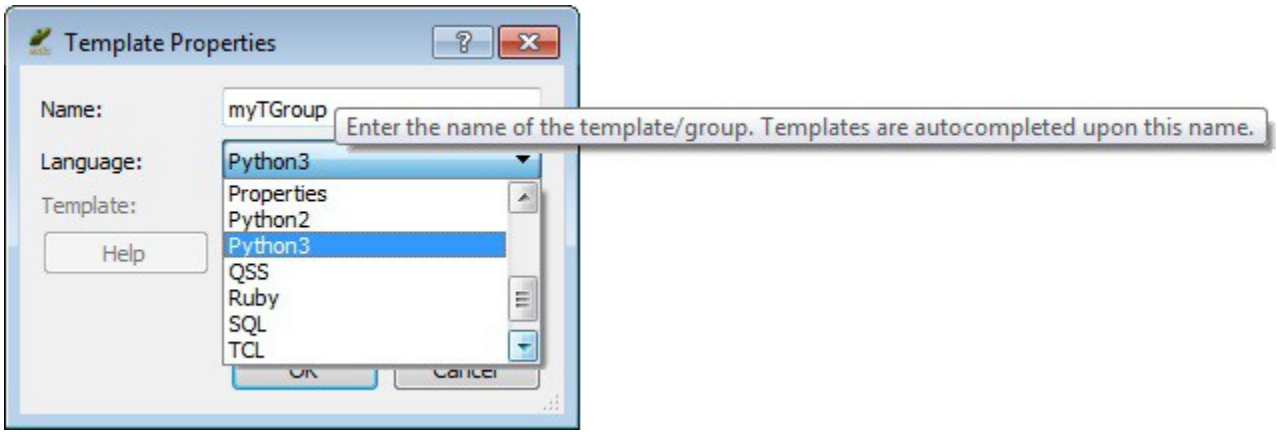
<.> A manageable "`Template Help`" instruction box, conveniently scrollable and with the text copy- pasteable, so to record and print it at will.


Template Help
Template Help
To use variables in a template, you just have to enclose the variablename with $-characters. When you use the template, you will then be asked for a value for this variable.
Example template: This is a $VAR$
When you use this template you will be prompted for a value for the variable $VAR$. Any occurrences of $VAR$ will then be replaced with whatever you've entered.
If you need a single $-character in a template, which is not used to enclose a variable, type $$(two dollar characters) instead. They will automatically be replaced with a single $-character when you use the template.
If you want a variables contents to be treated specially, the variablename must be followed by a ':' and one formatting specifier (e.g. $VAR:ml$). The supported specifiers are:
ml Specifies a multiline formatting. The first line of the variable contents is prefixed with the string occurring before the variable on the same line of the template. All other lines are prefixed by the same amount of whitespace as the line containing the variable.
rl Specifies a repeated line formatting. Each line of the variable contents is prefixed with the string occuring before the variable on the same line of the template.
The following predefined variables may be used in a template:
date today's date in ISO format (YYYY-MM-DD)
year the current year
project_name the name of the project (if any)
project_path the path of the project (if any)
path_name full path of the current file
dir_name full path of the parent directory
file_name the current file name (without directory)
base_name like file_name, but without extension
ext the extension of the current file
cur_select the currently selected text
insertion Sets insertion point for cursor after template is inserted.
select_start Sets span of selected text in template after template is inserted (used together with 'select_end').
select_end Sets span of selected text in template after template is inserted (used together with 'select_start').
clipboard the text of the clipboard
If you want to change the default delimiter to anything different, please use the configuration dialog to do so.
Copy Ctrl+C
Select All Ctrl+A
Close

-<>-

## Template (^) **Add Group...**

Designed to show a "`Template Properties`" control box where to create a *Template Group*, as the required container for actual Templates.

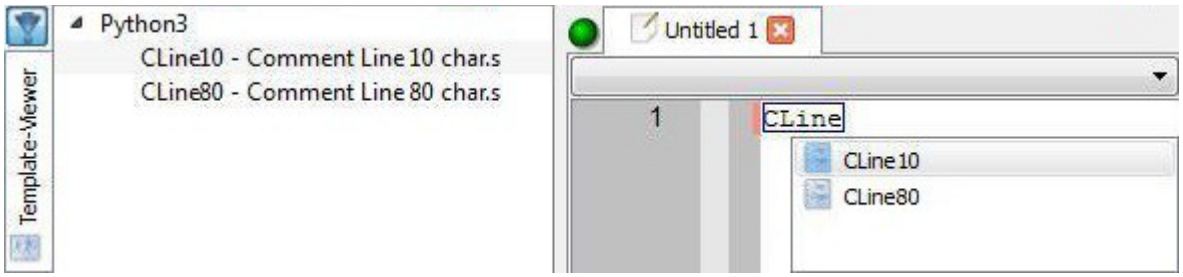



This is the same control box of the `Template (^) Add Entry...` command [*see*], but with the `Template` and `Help` controls here conveniently disabled.

<。> Template Groups are stored into disc as a XML file `<User>\_eric6\eric6templates.e4c`

### *Remark*

Template Groups named exactly after the related `Language`—e.g.: "`Python3`", instead of a generic "`myTGroup`" name—are intended to play a special role, as shown in the following example.




<。> Templates stored into a Group named exactly as the related programming language—such as here: `Python3`—can be inserted into a source text simply entering the Template's name, or even part of the name, followed by a trailing `<Tab>`. That is, instead of clicking on the usual `Template (^) Apply` context command [*see*].

– –

## Template (^) **Edit...**

Practically identical to the command `Template (^) Add Entry...` [*see*], just provided for acting onto an existing and selected Template, rather then a new one to be created.

– –

## Template (^) **Remove**

Designed to permanently remove a selected Template, or Group of Templates.

<!> Note that even this action to take effect requires a `(^) Save` action [*see hereafter*] and, therefore, could be discarded with a `Template (^) Reload` [*see hereafter*] entered before exiting Eric.

– –

## Template (^) **Save**

Designed to force writing from core memory to disc all Templates possibly changed during the current session, an action that would anyhow occur automatically at Eric exit. Command disabled whenever no change has been done to any Template.

<!> Undesired changes could be anyhow discarded before exiting Eric with a `Template (^) Reload` action [*see hereafter*].

– –

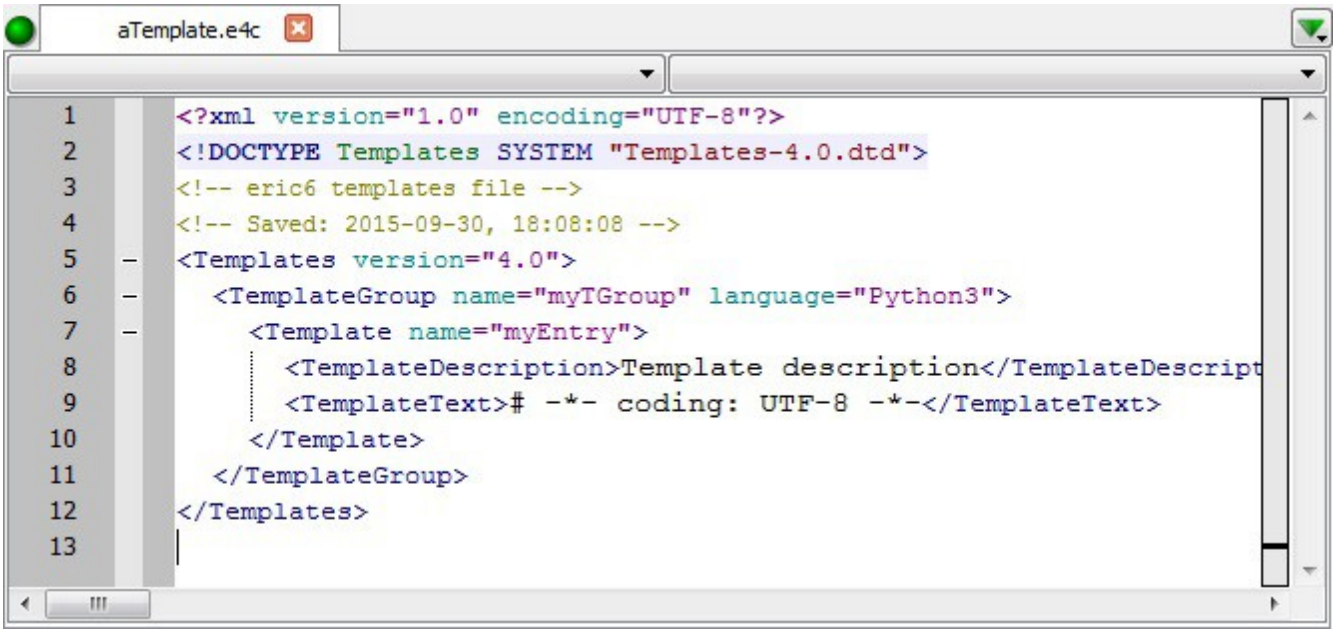
## Template (^) **Import...**

Designed to import an Eric Template collection "`*.e4c`" file, formerly exported with command `Template (^) Export...` [*see*]. The contents of the given file will be then copied and merged into the current `<User>\_eric6\eric6templates.e4c` file [*cf.: above* `(^) Add Group...` *command*]. Former contents unaltered.

– –

## Template (^) **Export...**

Designed to export the current Template collection into a XML "`*.e4c`" file, so to be then possibly imported into another Eric environment [*cf.:* `Template (^) Import...`].

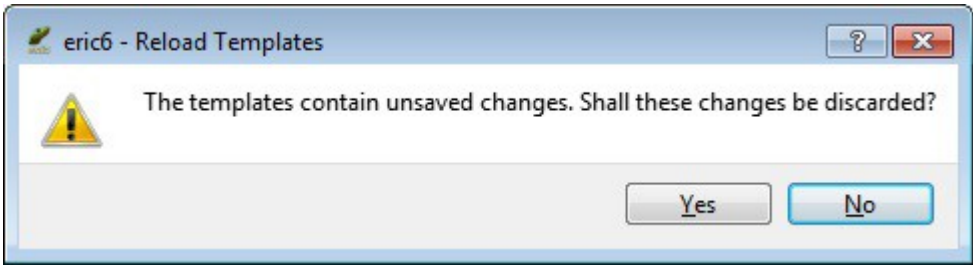



<.> Note that any such exported file can be conveniently opened and shown also on an Eric's text edit form.

– –

## Template (^) **Reload**

Designed to reload current Templates from disc into core memory. In case of unsaved changes you will be warned that this is an overwrite action that will reset the original conditions as recorded on disk.




<!> Which, of course, could be well used by purpose, as a way to discard undesired edit actions [*cf. above:* `Template (^) Save`].

### *Remark*

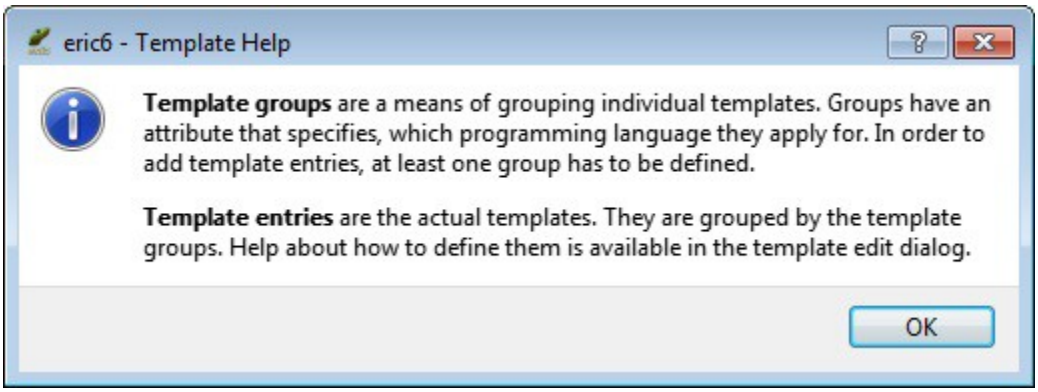
Note, in particular, that a `(^) Reload` action on:

- Groups or Templates just created, but non yet `(^) Saved`, will simply discard them from core;
- Groups or Templates created acting "directly" on their standard container, that is the XML disc file "`<User>\_eric6\eric6templates.e4c`", will simply bring them as they are from disc to core memory.

– –

## Template (^) Help about Templates...

Designed just to display such a "`Template Help`" info dialog box:




– –

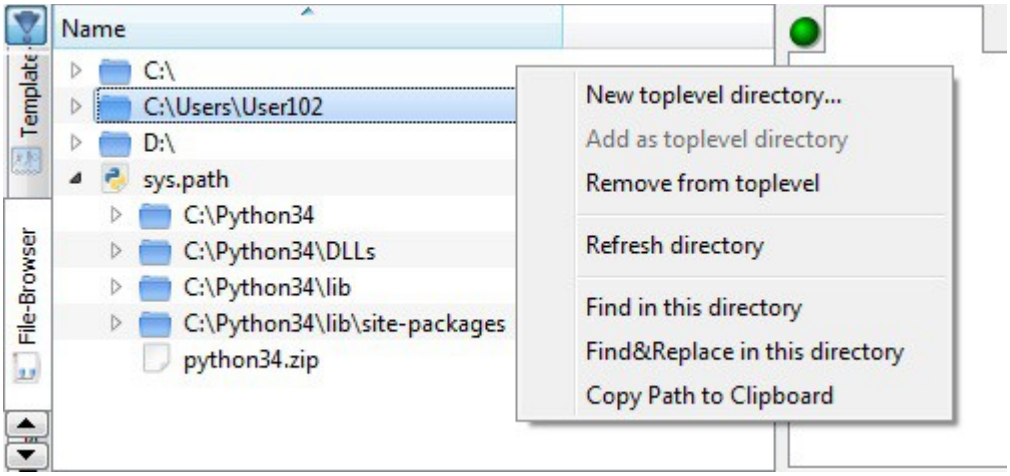
## Template (^) Configure...

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure Templates`" control box.

-<>-

## *L-Pane*: File-Browser

A general purposes file-browser[140], with also such a Python-oriented "`sys.path`" expandable view [*verify*], here handy available within Eric.




### *Remark*

Among the possible differences between this Eric file browser and that one in use in your operating system, there is the alphabetic ordering. For instance here you'll find such a "`_XName`" not before but *after* a "`XName`".

#### *Command List*

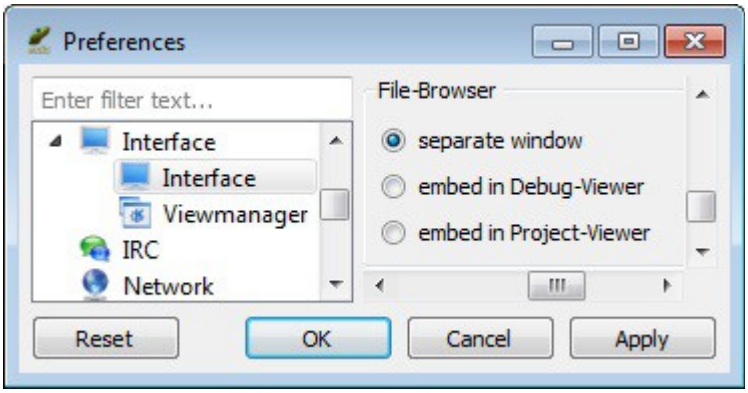
`New Toplevel Directory...` `Add as Toplevel Directory` `Remove from Toplevel`
`Refresh Directory` `Find / Find&Replace in This Directory`
[*ref.:* `Tab (^)`, *in:* {2.Central}] `Copy Path to Clipboard`

– –

## Alternative "File-Browser" Location

It's here worth noting that, as with other Eric controls [*cf.:* Alternative Text "List View" Form, *in:* {2.Central}], also this File-Browser could be configured with command `Settings > Preferences... – Interface > Interface, Configure User Interface, File-Browser` [*see*] so to be located elsewhere, instead of into a "`separate [tab] window`" on the L-Pane, as here considered.

140Whereas the not-so-different Project-Viewer's "`Sources`" browser, here above [*see*], is there to offer a dedicated view on a currently open Project.




For instance it could be embedded into the Project-Viewer on this same L-Pane, or into the Debug-Viewer on the Auxiliary Right Pane, with such possibly new condition that will be then enabled at the next application startup.

### *Viewpoint*

<。> Note that in this case too, as throughout all this Report, we'll make reference to the initial default condition only, for obvious reasons of simplicity.

– –

## File-Browser (^) **New Toplevel Directory...**

Designed to bring here in display[141], as a "Toplevel Directory", another directory of your file system, as for instance the `C:\Python3X\Lib\site-packages\eric6`; so to get an easy first-hand look into it.

Inverse action performed with next `(^) Remove from Toplevel` command [*see*]. Original directory otherwise unaffected.

– –

## File-Browser (^) **Add as Toplevel Directory**

Similar to the former "`New Toplevel`" command [*see*], but with the selection of the directory executed using this same `File-Browser`. Original directory otherwise unaffected.

– –

141That is: no creation is here involved, as the *New* keyword could lead you to assume.

## File-Browser (^) **Remove from Toplevel**

Designed to remove the “Toplevel” display condition of a directory. That is, command designed to act as the inverse of former two commands [*see*]. Original directory otherwise unaffected.

– –

## File-Browser (^) **Refresh Directory**

Just to assure an updated display of the selected directory, as possibly modified by external actions.

– –

## File-Browser (^) **Find in This Directory**

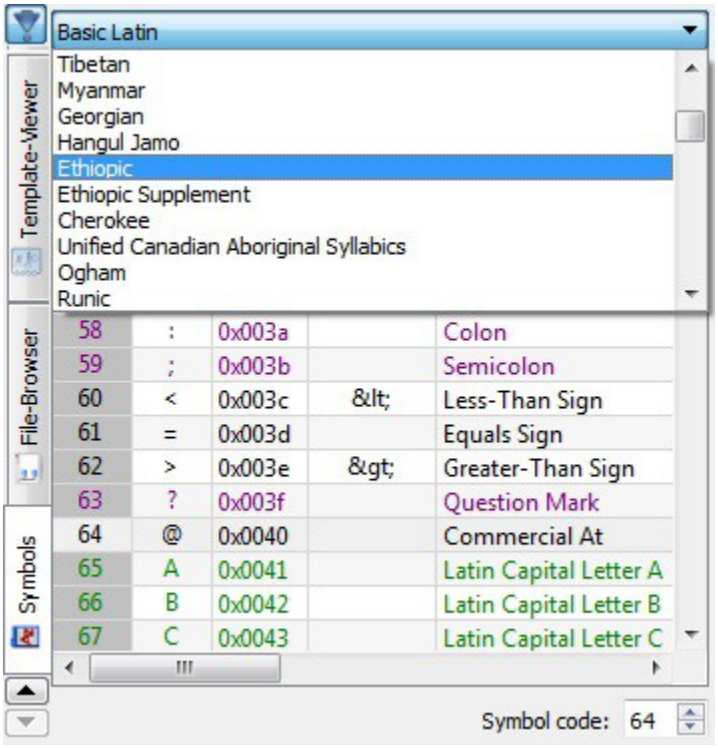
## File-Browser (^) **Find&Replace in This Directory**

Designed to grant also here such powerful file editing tools, usual with Unix-like environments.

–<>–

## *L-Pane*: **Symbols**

When you need to know code and shape of such a typographic symbol set as the usual "Basic Latin", or the special "Mathematical Operators", or the not so usual "Ethiopic"[142], here is where to go.




A double-click action on a selected symbol will enter it at the current text insertion pointer position. That said, a tool self-explanatory enough not to require any further description.

– = –

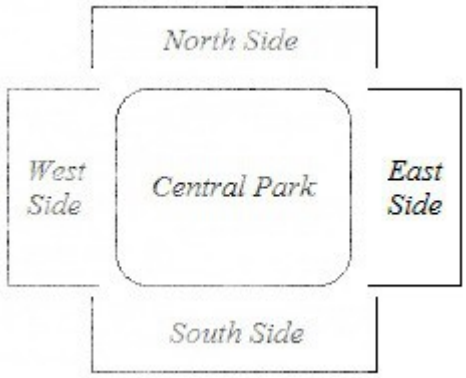
142Ethiopia is where ancient Gods of the Greek mythology used to go in holiday, and also where we've been lucky enough to live and work, for some time.

# {5.East} Eric Window – *East Side*

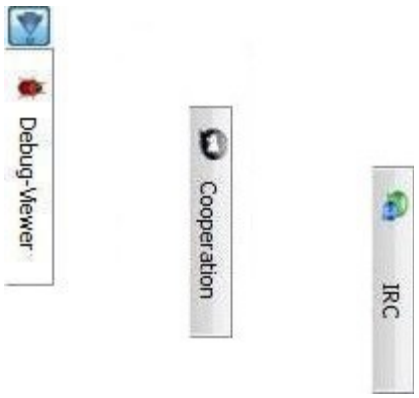
*S-PM 160400*

## Auxiliary Right Pane

Here we are at the "East Side" area of the Eric Application Window [*cf.:* Eric Window Map*, in:* {Map}*, at the end of this* Report].




An optional pane positioned at right on the Eric window[143], hosting a tabbed form comprising the following vertical tabs, ordered from top to bottom and described in detail on subsequent sections: `Debug-Viewer, Cooperation, IRC`



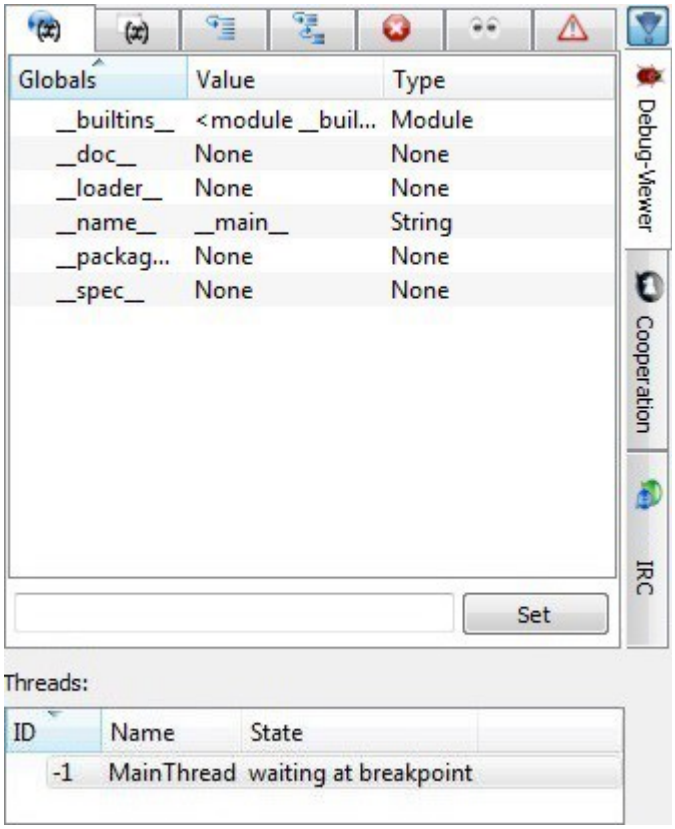

Optional in the sense that its presence can be controlled with the main menu command `Window > Right Sidebar`, selected with `Window > Edit Profile / Debug Profile` commands and configured on the "`Configure View Profiles`" control box of command `Settings > View Profiles...` [*see*].

143"R-Pane" hereafter, for short.


Plus, when in display, also with the usual toggle-control button for the automatic collapsing mechanism.

– –

## *R-Pane*: **Debug-Viewer**




This the `Debug-Viewer` tab, comprising two forms:

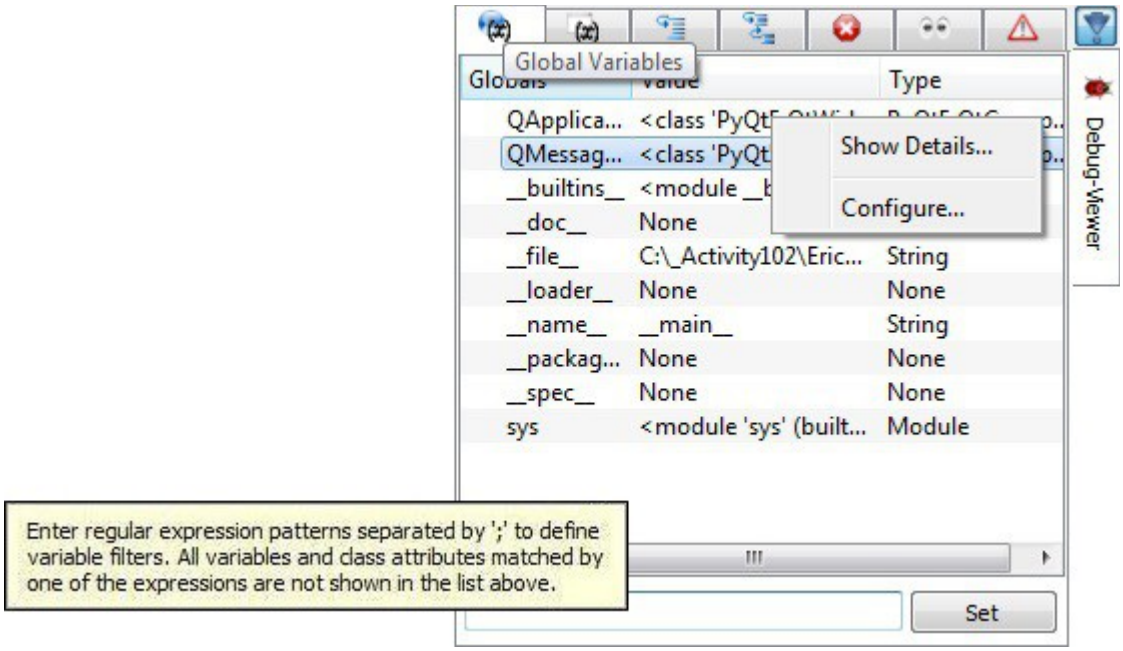
`Debug-Viewer` A tabbed form, comprising the following top-side horizontal tabs, ordered from left to right: `Global Variables`, `Local Variables`, `Call Stack`, `Call Trace`, `Breakpoints`, `Watchpoints`, `Exceptions`; hereafter described orderly, in detail.

`Threads` A table meant primarily for debugging Python multithreaded programs written making use of the "`threading`" Python module[144], but here shown also with the debugging of single thread programs.

– –

## *Debug-Viewer:* Global Variables

This "`Global Variables`" form results fully activated when an actual Python debug execution is under way.




### *Specifications:*

`Set` Button to exclude some item on the list [*see:* `Help > What's This?`]

– –

144An advanced programming technique about which nothing can we here say, as we haven't matured any specific experience [*ref. URL:* http://pymotw.com/2/threading/].

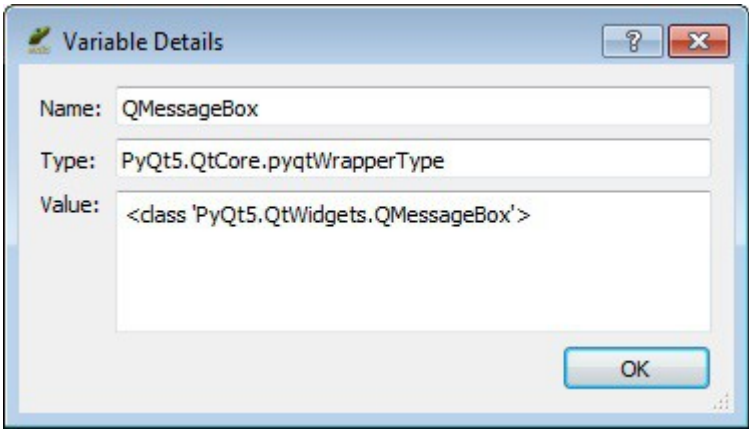
### *Command List*

`Show Details...` `Configure...`

– –

## Global Variables (^) **Show Details...**

Designed to show a "`Variable Details`" dialog box where to display all details of the selected Global Variable.




A double-clik on that variable will have the same result.

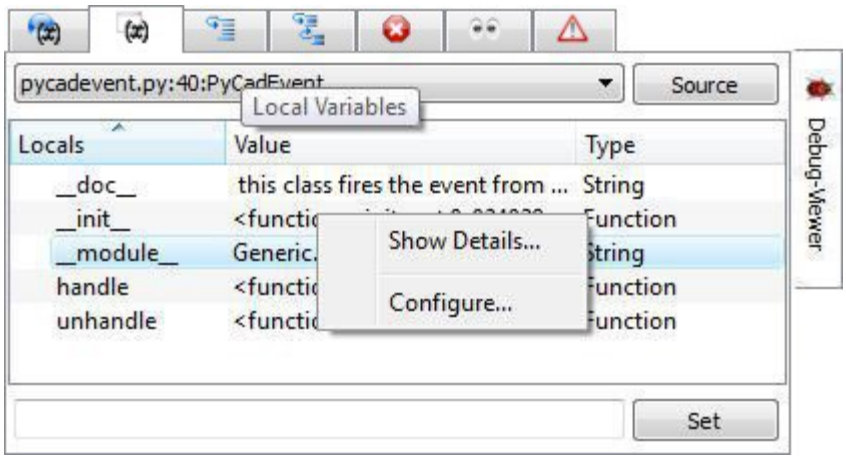
– –

## Global Variables (^) **Configure...**

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure general debugger settings`" control box.

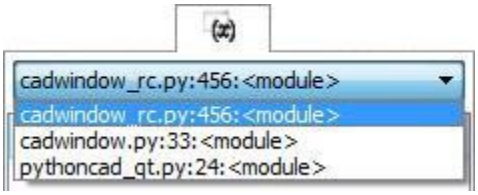
– –

## *Debug-Viewer:* **Local Variables**




This "`Local Variables`" form operates as the former "`Global Variables`" form [*see*], plus with:

`<module>` A "`source file:line#:`" drop-down list of the program's call stack, that is




a list of the program steps executed up to the current point [*verify*].

`Source` A button to synchronize the item selected on the list with the related text edit form. A button that will be not shown in case of automatic synchronization enabled with command `Settings > Preferences... - Debugger > General, Local Variables Viewer, Automatically view source code` [*see*].

– –

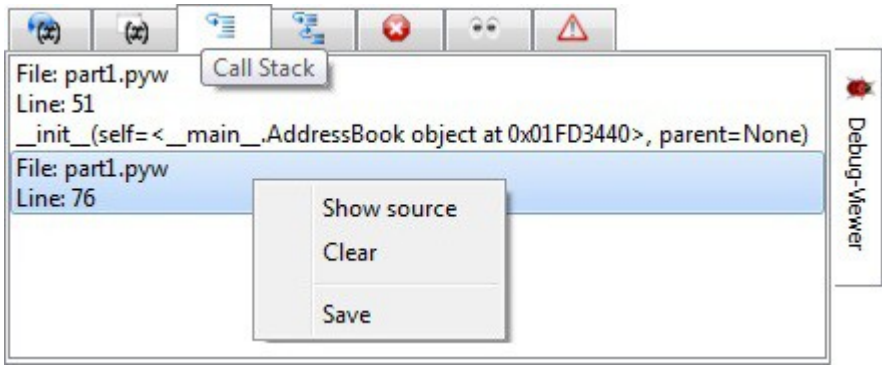
### *Command List*

[*ref.:* `Global Variables (^)`] `Show Details...` `Configure...`

– –

## *Debug-Viewer:* Call Stack

Designed to display source stack info of possibly nested execution steps.




### *Command List*

`Show Source` `Clear` `Save`

– –

### Call Stack (^) Show Source

Designed to synchronize the Source Text Edit form with the item selected on this `Call Stack` list.

– –

### Call Stack (^) Clear

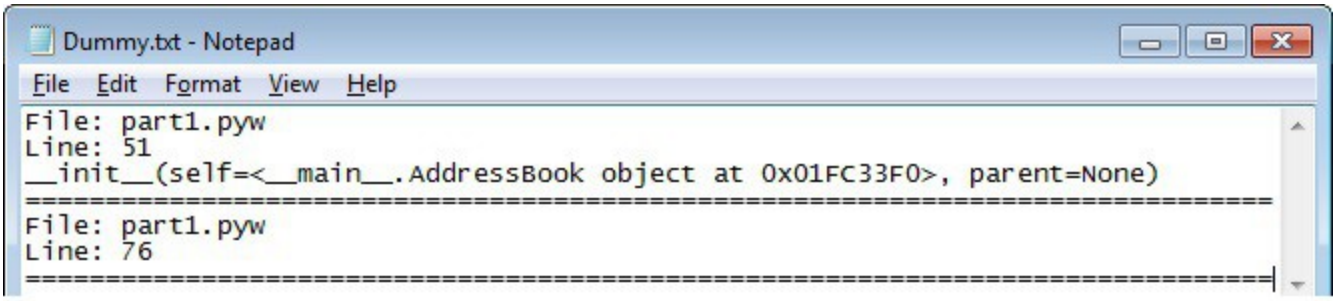
Designed to clear all info possibly listed on this form.

– –

### Call Stack (^) Save

Designed to record into a text file[145] all info currently listed on this form.

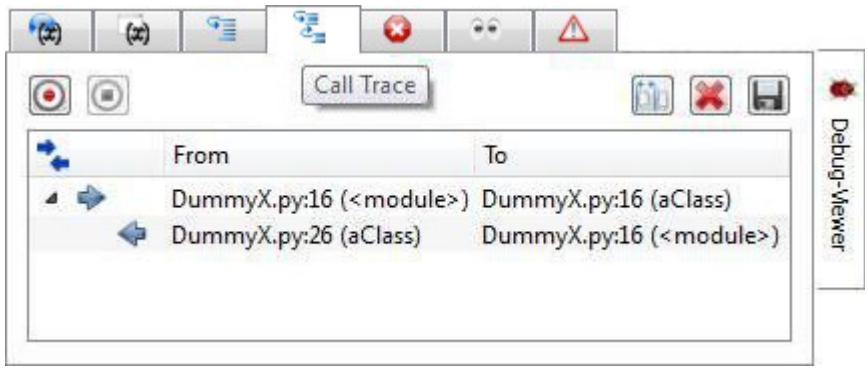
145<~> Here too we'd suggest a better choice for the initial default directory, as the Project Directory instead of the Python installation's one, as it is now with current rev. 6.1 [*cf. hereafter:* `Call Trace`].




– –

## *Debug-Viewer:* **Call Trace**

Designed to intercept and show the function calls and returns occurring during a program's execution. To be used in combination with a `Start > Debug` execution and the convenient positioning of breakpoints, so to activate the tracing precisely after the desired execution point.




***Controls*** [*all actions through form controls, no context menu needed*]

/ Start / Stop tracing. Note that tracing affects the executing performance, as it drains some computing resources.

To resize the table columns width according to the contents

To reset and clear tracing data

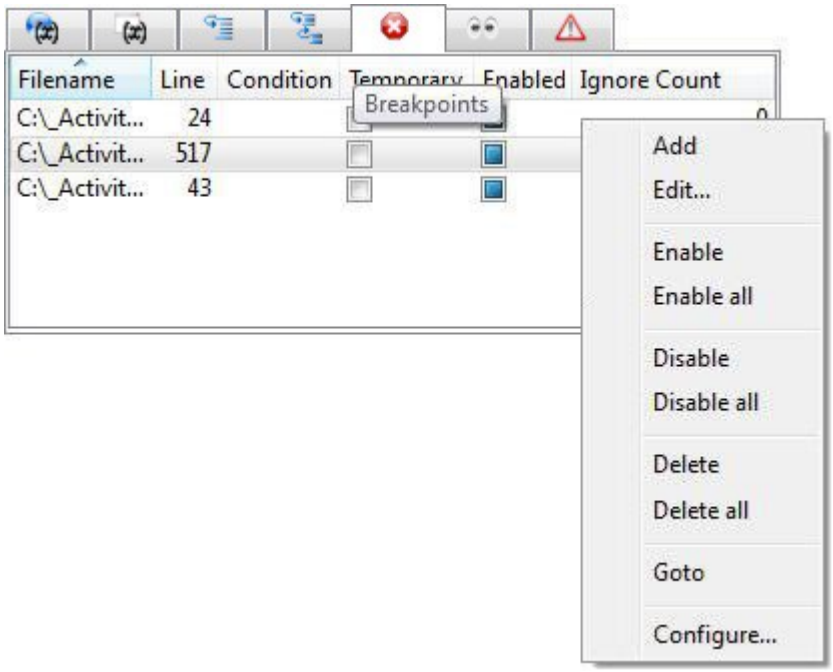
To save current tracing into a text file[146]

– –

146<~> Here a better choice for this default directory would be the Project's, instead of the Python's installation directory.

## *Debug-Viewer:* **Breakpoints**

Context management and use of usual Eric debug breakpoints [*see:* Debug Command Menu, *in:* {1.North}].




### *Command List*

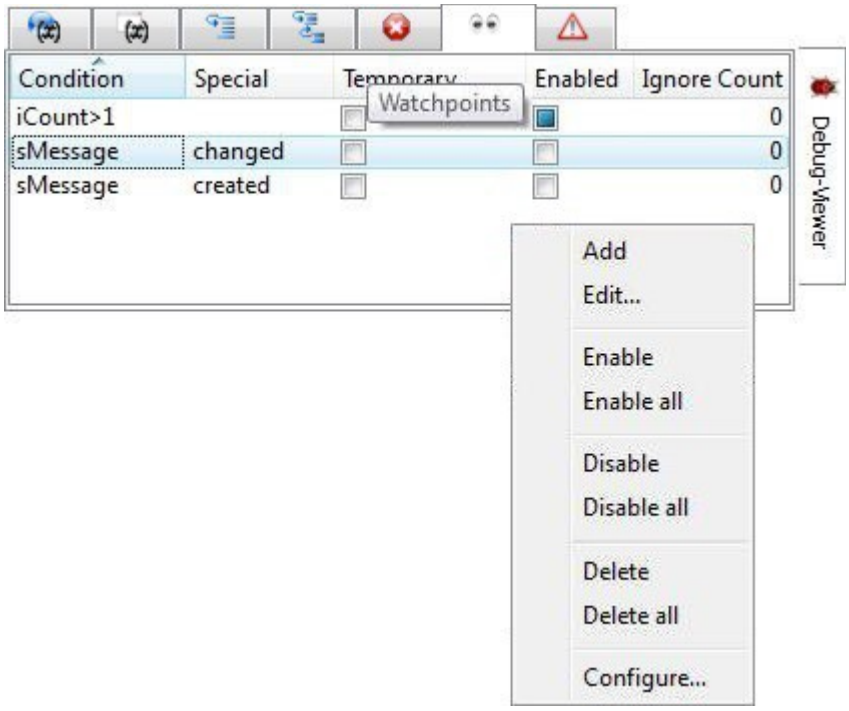
| | | | |
|---|---|---|---|
| [*ref.:* `Debug > Toggle Breakpoint, …`] | `Add` | `Edit...` | `Enable` |
| [*ref.:* `Debug > Toggle Breakpoint, …`] | `Enable All` | `Disable` | `Disable All` |
| [*ref.:* `Debug > Toggle Breakpoint, …`] | `Delete` | `Delete All` | `Goto` |
| [*ref.:* `Global Variables (^)`] | `Configure...` | | |

<。> A set of commands assumed of immediate comprehension, with no further explanation required.

– –

## *Debug-Viewer:* **Watchpoints**

Eric Watchpoints, an advanced form of breakpoint bound to an execution condition, instead of simply to a source code line. They operate during debug execution and are to be defined and managed by means of the hereafter context commands [*notably:* `Add` *and* `Edit...`].




### *Command List*

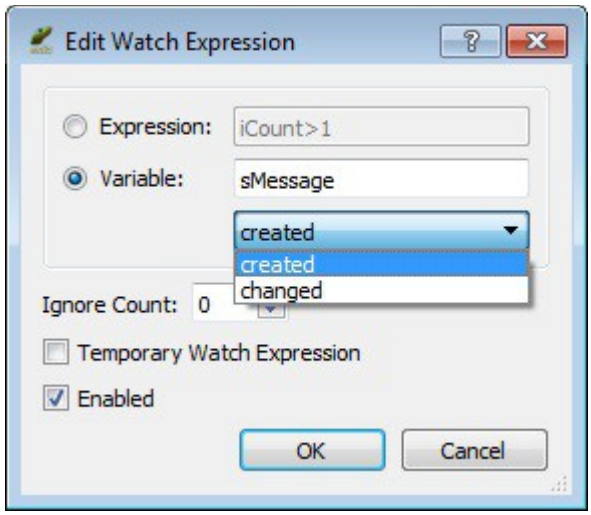
`Add` `Edit...`
[*self-explanatory*] `Enable` `Enable All` `Disable` `Disable All` `Delete` `Delete All`
[*ref.:* `Global Variables (^)`] `Configure...`

– –

## Watchpoints (^) **Add**

## Watchpoints (^) **Edit...**

Designed to show an "`Edit Watch Expression`" control box, where to define / edit a "Watch" break condition for a debug execution.




*Specifications:*

`Expression` Field where to enter the desired "Watch" condition, such as this one: `iCount>1`, based upon the value possibly assumed by a variable during the program's execution, in debug mode. A condition in alternative with next "`Variable`" [*see*].

`Variable` Field where to enter a variable for which to intercept the selected one of these two special "Watch" events: `created` / `changed`
A condition in alternative with former "`Expression`" [*see*].

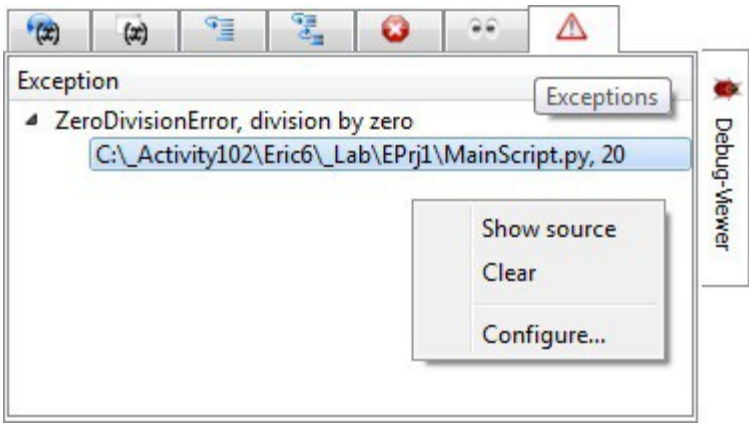
`Ignore Count` Condition to be ignored that many times, typically useful in debugging execution loops [*cf.:* `Debug > Edit Breakpoint...`]

`Temporary` Condition to be executed just one time, then disabled [*cf.:* `Debug > Edit Breakpoint...`]

`Enabled` To enable / disable this Watch break condition

– –

# *Debug-Viewer:* **Exceptions**




## *Command List*

```
     Show Source            Clear
[ref.: Global Variables (^)] Configure...
```

– –

## Exceptions (^) **Show Source**

Designed to synchronize the Source Text Edit form with the item selected on this `Exceptions` list.

– –

## Exceptions (^) **Clear**

Designed to clear all items possibly listed on this `Exceptions` form.

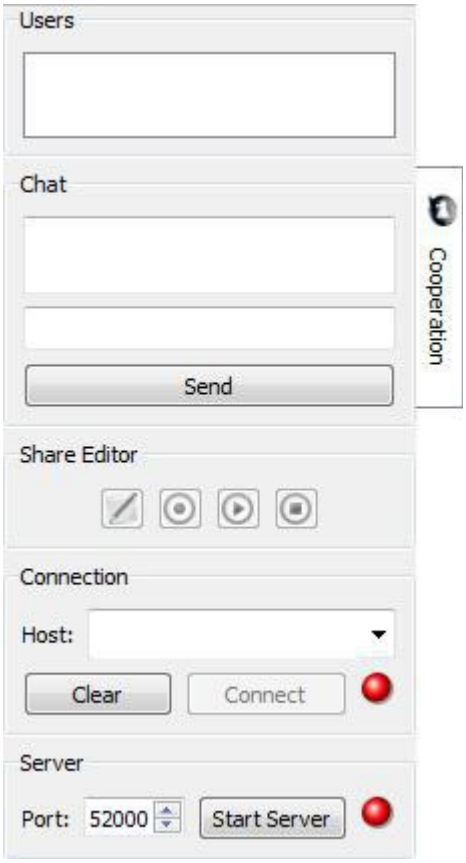
-<>-

## *R-Pane*: **Cooperation**

A tool aimed at permitting the Eric users connected in local network, or in the Internet, to work in team as Cooperation Participants, through:

- Simple “Chat” messages interchange;
- Cooperative source text editing.

This the `Cooperation` control form, as in its initial appearance.




### *Remark*

As a convenient way for testing this feature, a Cooperation can be established between two Eric instances running on the same computer. Note that, to this purpose, the `Settings > Preferences... - Application, Single Application Mode` check-box must be unchecked [*see*].

– –

### *Viewpoint*

<。> Then, to make real use of this feature in Internet, you must have enough control over the connection so to have your host actually "visible"[147] in the Internet.

Being this not our case, hereafter we'll refer exclusively to a "local" Cooperation test set-up, based upon different Eric instances operating on the same computer system.

– –

## Cooperation Controls, and Functional Description

`Start Server` Button to activate the Eric built-in cooperation server. Upon successful activation the related red LED will turn green.

– –

`Port` Numeric parameter playing the role of a Port number, required to identify a specific "extension" (i.e.: sub-address) within a single computer identified by a unique IP address in a TCP/IP protocol network.

Port numbers are used to convey data to the "right" application among the possibly many ones concurrently running in the same computer. It's a number in the range of 0÷65535, conventionally divided into the three sub-ranges[148] called: *well-known ports* 0÷1023, *registered ports* 1024÷49151, and *dynamic* or *private ports* 49152÷65535[149].

Hereafter an example of two Eric instances cooperating on the same computer.


Note that the Port number chosen for the second Eric instance on the same computer is automatically proposed different from the first one.

– –

147"Visible" in the Internet means that your router must be configured to relay a port to your Eric machine. In addition to that, your router should have an entry with a Dynamic DNS provider, or else you have to tell your counterpart what is the external IP-Address of your router.

148A convention overseen by the Internet Assigned Numbers Authority (IANA).

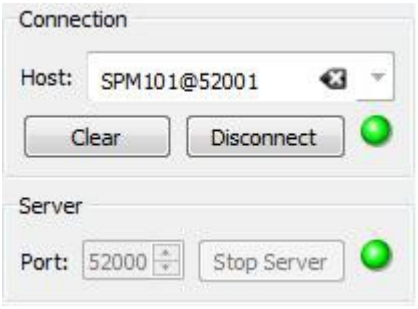
149<~> Note that the initial Port number 52000, here suggested as default, is prudentially chosen not within the set of *registered* [*as the unfortunate original Eric ver. 6 default value 42000*], but within that of *private*, typically used as "ephemeral" ports [*see:* `Settings > Preferences... – Cooperation, Server Port`].

`Host` A [<~> *rather misleading*] label for the "**`Host@Port`**" parameter to be entered here, so to cooperatively connect the current Eric instance to another cooperating instance.

*Specifications:*

| | |
|---|---|
| `Host` | Computer Name of the target Eric instance, e.g. an explicit: `SPM101` or a parametric: `localhost` |
| `@` | Separator |
| `Port` | Port number of the target Eric instance, e.g.: `52001` |

Hereafter an example of connection request with Host parameter: `SPM101@52001` which, in this case, is equivalent to: `localhost@52001`




As a consequence of this connection request, the Eric target instance will receive such a dialog box:


Where the requesting instance is identified with a `<User Name>@<Computer Name>` code, written in a suitable format, different from the "`Host@Port`" afore mentioned, so to conveniently identify the caller in the system.

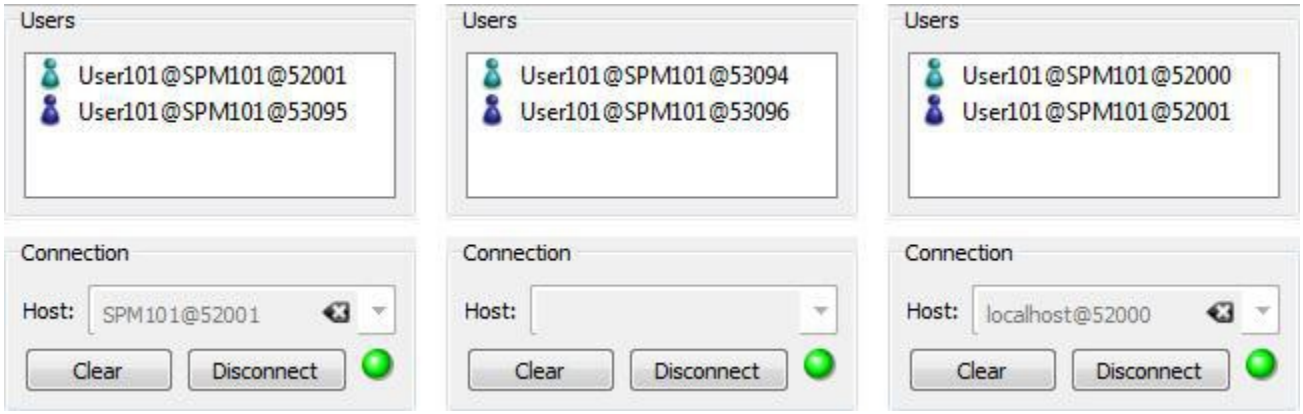
## *Remark*

Note that here there is no way to know which are the other Eric instances possibly available and ready to accept a connection request. Therefore, for activating such a cooperation, you need to preliminarily rely upon another communication channel[150], such as e-mail or voice.

– –

150<~> Not a negligible trouble, we'd dare say.

`Users` It's the box where you see whomever else you are currently connected with. As an example, hereafter you see how three different Eric instances are mutually informed about the respective cooperative connections.




– –

*It's here worth observing that:*

- Each Eric instance can activate only one connection, and only if not already connected.
- Each new connection adds up to all the existing others, into a peer-to-peer network.
- A pop-up context menu is here available, simple enough not to require any further explanation[151].


- Connected nodes are listed on the "`Users`" area with an identifying code which has got a [*curiously*] different format from the "`Host@Port`" as it is required to activate a connection [*cf.: above "`Host`" fields*].

– –

151<~> Anyhow here we doubt that *any* of the participant be actually enabled to "Kick" or "Ban" *any others* [*verify*].

## *Viewpoint*

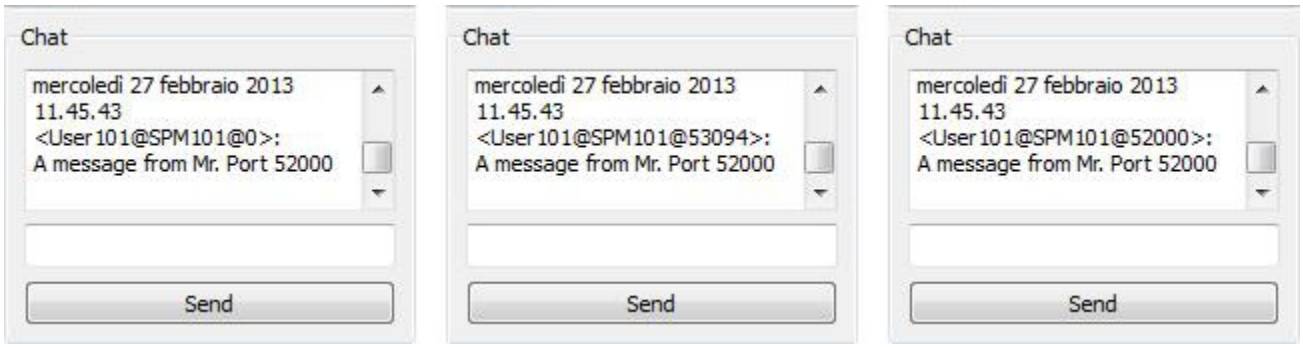
<~> The above "[*curiously*]" clause is to express our (mild) sense of perplexity that we'd like here to justify with this example:

- If you happen to be Mr. Host `SPM101@52001`, asking to connect Mr. Host `SPM101@52002`, …
- your target companion will receive your connection request as coming from as Mr. `User101@SPM101`
- Then, if accepted, you'll see your connected companion as a: `User101@SPM101@52002`, …
- and your companion will see you as, say: `User101@SPM101@54596`.

Not two of these codes expressed with the same format. We are fairly sure that all these different ways for identifying precisely the same subject have got some solid justification, nevertheless we can't help expressing our (mild) perplexity about all such different formats used for referring precisely to the very same item.

– –

`Chat` When connected, Eric instances can "Chat" as shown in the example hereafter, where you see how "`A message`" sent by "`Mr. Port 5200`" is received by all currently cooperating participants, sender included.

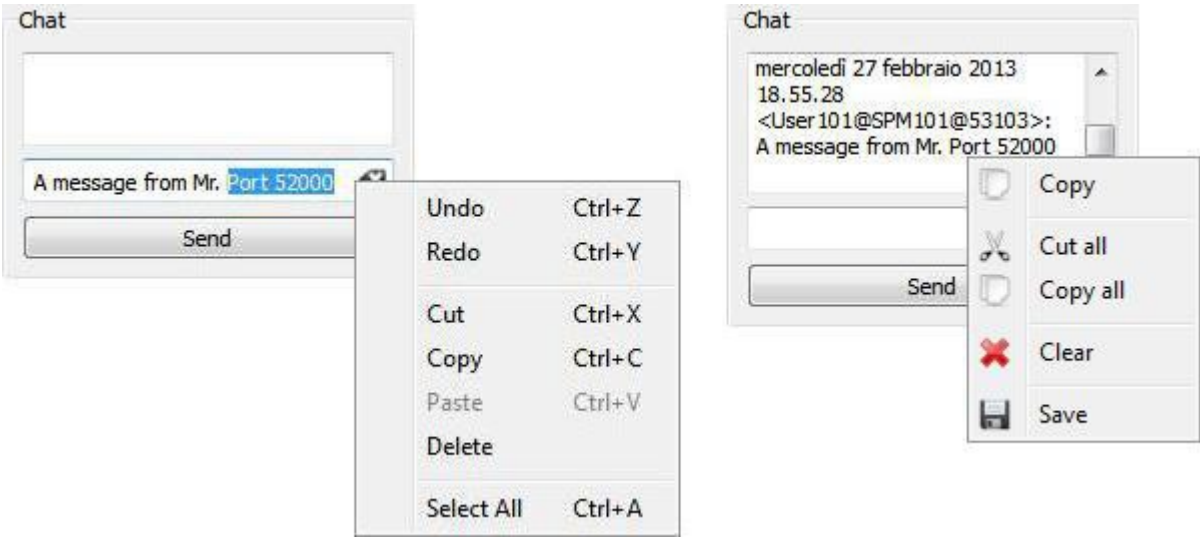



– –

### *It's here worth observing that:*

- Into the "`Chat`" field the texts can be inserted interactively or pre-composed elsewhere, and then there pasted. <!> Test-messages of up to 2000 characters has been sent successfully.
- A <CR> entered interactively into the "`Chat`" text field works as the `Send` button, whereas <CR> characters entered as part of a pasted text are here accepted and sent as any other character.

- In the "`Chat`" area the sender is [*curiously*] identified with a code which is different from the original "`Host@Port`", and also different for each receiver [*cf. above* `Host` *and* `Users`].
- Two context menus are here available, simple enough not to require any further explanation.




– –

`Share Editor` Cooperative source text editing feature:


Typical cooperative procedure, requiring the same Project name and the same file name on all the cooperating Eric instances, possibly operating in network and on different discs.

*Steps:*

Shared *Status* toggled on for all cooperating instances, so becoming: ;

Shared *Edit* session started in one "master" instance. In all other instances the same file is automatically turned "read only" `(ro)`:


 *Send* control, for pushing the shared edit changes from the current master instance to all other instances connected and in cooperative edit;

 *Cancel* control, for the current master edit session, so to possibly let another instance to become master.-

-<>-

## *R-Pane*: **IRC**

Here you have an IRC (Internet Relay Chat) *Client* integrated into Eric, ready available to connect you "live"—i.e.: in real-time—with such multi-user, multi-channel chatting system as a whole and, in particular, by default, with the specific Network community of Eric users.


Once connected to the Internet, available IRC *Servers* can be reached and utilized with the tools as hereafter described.

– –

## IRC Controls, and Functional Description


To select an IRC *Network*, here defaulted to `Freenode.net` as used by the Eric community people. See this  button[152] to manage (add / edit) the set of IRC Networks here available.

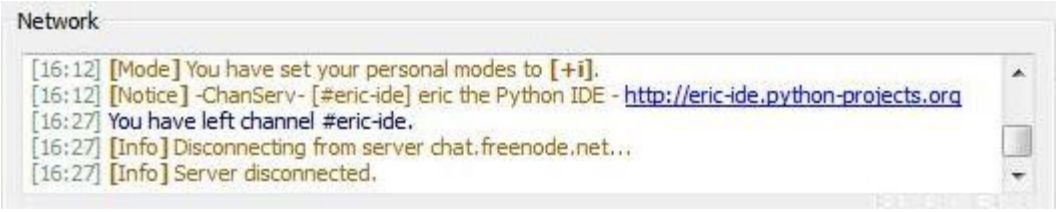
152Detailed description of this feature is here considered beyond the scope of this Report [*see:* Scope of this Report, *in:* {0.Lead}].

Note that also the command `Settings > Preferences... - IRC` will offer you a `Configure IRC` control box, but it's essentially meant for cosmetic, not functional aspects [*verify*].

– –

To connect / disconnect the selected Network.

Once connected, here is where the Network messages are shown:

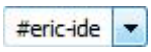



– –

To set / reset your "AWAY" user status, so to possibly tell you are connected, but not currently paying attention to IRC.

Note that this same icon will be shown, brightened, on the Channels user list at right of each nickname currently in that AWAY status [*see image hereafter*].

– –




To set and select which the IRC *Channel* (i.e.: IRC session) to join, here defaulted to the `#eric-ide`, for the community of Eric users.

– –


To select the *Nickname* you wish to be known by in the Network, here defaulted to the computer User Name.
Note that:

- This drop-down list is enabled only when a Network connection has been established, not before, as it is with the `Channels` list[153].

153<~> And, frankly, as we'd expect.

- Custom Nicknames can be preset via the "`Edit Networks`"  feature, "`Edit Identities...`" button, "`IRC Identities`" control box [*see*].


## *Viewpoint*

<~> And that to our surprise, we should say, as we wouldn't expect the Nickname classified as a network's property.

– –

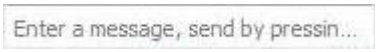
Buttons to join / leave current Channel.


– –

Once joined, here is where are shown the active Channel user Nicknames:

and the related messages interchanged:


– –

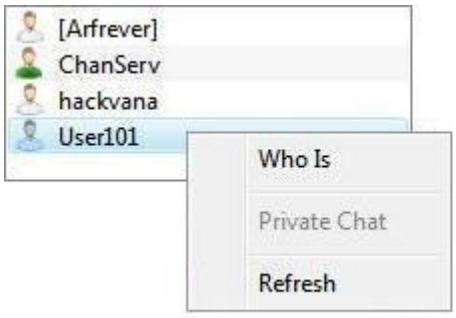


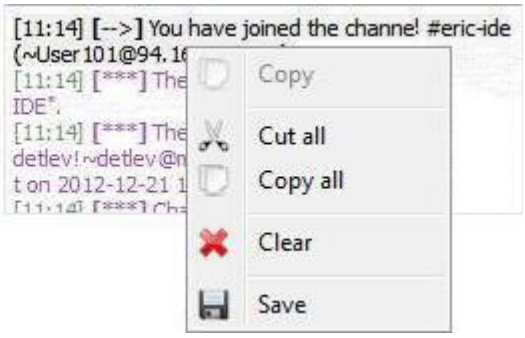

This the field where to enter IRC messages.

– –

IRC Channels have *Topics* of conversation, e.g.: "`eric - The Python IDE`" [*see*]. This control button turns available only for "`channel operators`" users, that is users with the permission of changing the current Topic.

Plus a couple of pop-up context menus that are here available, simple enough not to require any further explanation.




The context command "`(^) Private Chat`", when enabled, is to open a one-to-one chat Channel, instead of the usual one-to-all. Disabled for the current user and for users in AWAY status.

– = –

# {6.Trail} Setup and General Management

*S-PM 160400*

Reference section for setup and general management of Eric (6) Python IDE, onto the target environment here considered; that is: host operating system Windows 7, or Windows Vista Ultimate [*see:* Essentials, *table* Version Reference, *in:* {0.Lead}].

– –

### *Remark*

<!> Of all the possible setup combinations compatible with Eric ver. 6—that is: Python 3 or / and 2, with PyQt 5 and / or 4, along with all other related add-ons, tools and accessories [*see hereafter:* Optional Packages, *and* Other Optional Packages], as variously compatible with the cited versions — here we have chosen to consider, test and describe only the most recent base configuration, that is:

Python 3 with PyQt 5

A selective choice dictated by practical reasons of feasibility, sure that all users having to make a different choice will find no difficulty in extending this description to their specific situation and needs.

– –

## 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 standard system—as it may be with other platforms, such as Linux-based Ubuntu or Mac OS X—, therefore they must be collected from different sources, as hereafter described in detail.

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

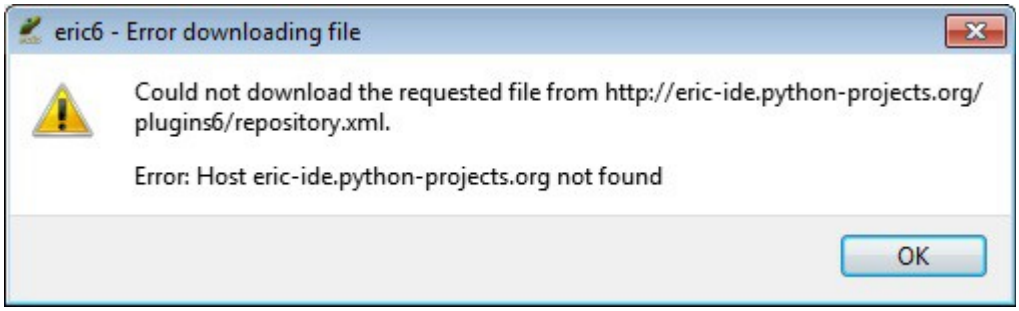
*Required* and *Optional*.

– –

### *Active Internet Connection as a Prerequisite*

An active Internet connection is currently factory-preset as an environmental prerequisite, so to grant possible automatic updating of the very Eric program and some other resources.

The absence of an active Internet connection available during the Eric execution may result in such an "`Error downloading file`" condition:




It is anyhow a user choice whether to keep such factory-default condition or else modify the related "`Check for updates`" options, via `Settings > Preferences...` [*verify*]; specifically for: `> Application, Configure the Application` and `> Plugin Manager, Configure Plugin Manager`.

– –

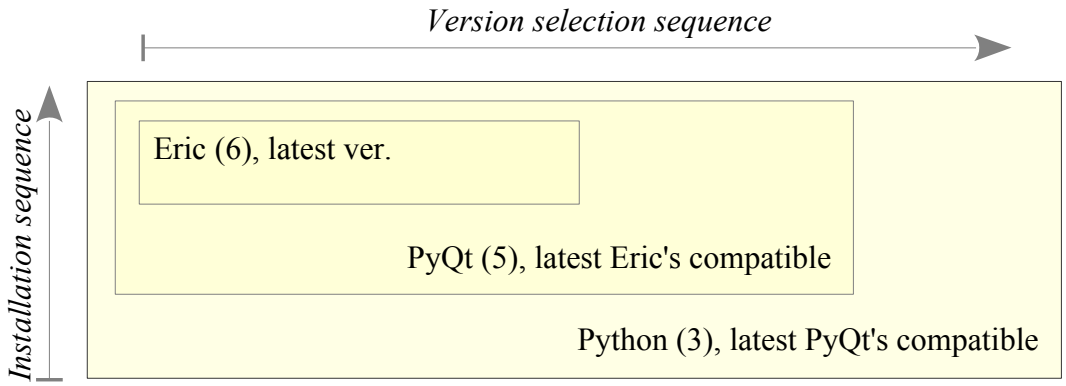
## *Required Packages*

*1/2]* Python Main Programming Language Interpreter, current ver. 3;

*2/2]* PyQt GUI Library, current ver. 5;

Eric Chief application, current ver. 6.-

You'll find related operative details on subsequent sections, ordered in the precise installation sequence as required by Eric. But it's anyhow necessary to be here immediately aware of the general criteria you should follow in choosing each package's version, and in which sequence to install each of them into you computer system; and why.




– –

### *Version Selection Sequence*

Reason for this precise sequence is, obviously, to assure the overall IDE compatibility.

i. Eric (6) latest revision, to begin with; e.g. [*1st quarter 2016*]: 6.1.*n*

ii. then PyQt (5) most recent revision *Eric compatible*; e.g.: 5.5

iii. last Python (3) most recent revision *PyQt compatible*; e.g.: 3.4 — which, by the way, is not the latest available in absolute, that is: Python ver. 3.5, for which, at the moment [*1st quarter 2016*], there isn't yet a compatible PyQt (5) version.-

### *Installation Sequence*

i. First Python (3);

ii. then PyQt (5);

iii. last Eric (6).-

Reason for this precise sequence is that, this way, each item can—*and should*—be immediately and independently tested before installing the next one, so to grant the overall functionality.

– –

## *Optional Packages*

What here follows is just a very limited and uncommitted example of Eric optional packages, being this an area substantially left to the user's free initiative and needs.

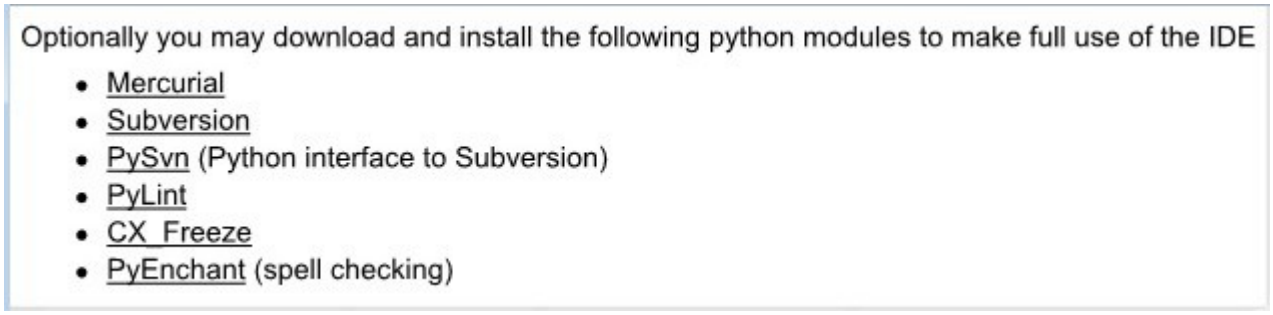
| | |
|---|---|
| Ruby | Another Eric compatible programming language |
| PyEnchant | Spell-checking library for Python |

More info and guide-lines can be found at command `Settings > Show External Tools`, `Plugins > Plugin Infos...` [*see: in* {1.North}] and at the following section "Other Optional Packages".

– –

## *Remark*

<。> Then also a look at the current "`Eric Python IDE - Download`" web page [*ref.:* What & Where in the Internet, *at* Foreword *sec.*] may be of help to further illustrate this subject.




– –

## *An Example*

Even before going to what here is the main subject, that is the Eric's "Setting of Required Packages", let's have just a preliminary look at an example of how to deal with a typical Optional Package, certainly not mandatory, simply rather useful; that is the spell-checker toolkit PyEnchant [*cf.: command* `Extras > Check Spelling...`].

## *Where*

PyEnchant, found at URL: `https://pythonhosted.org/pyenchant/`
There defined as: "a spellchecking library for Python, based on the excellent Enchant library."

## *What*

For convenience, Windows users are provided with a pre-built installation program which can be used to install PyEnchant: `pyenchant-1.6.6.win32.exe` [154]

## *How*

Declared prerequisite: Python 2.6 or later — with "later" that we happily assumed including also Python 3 compatibility.

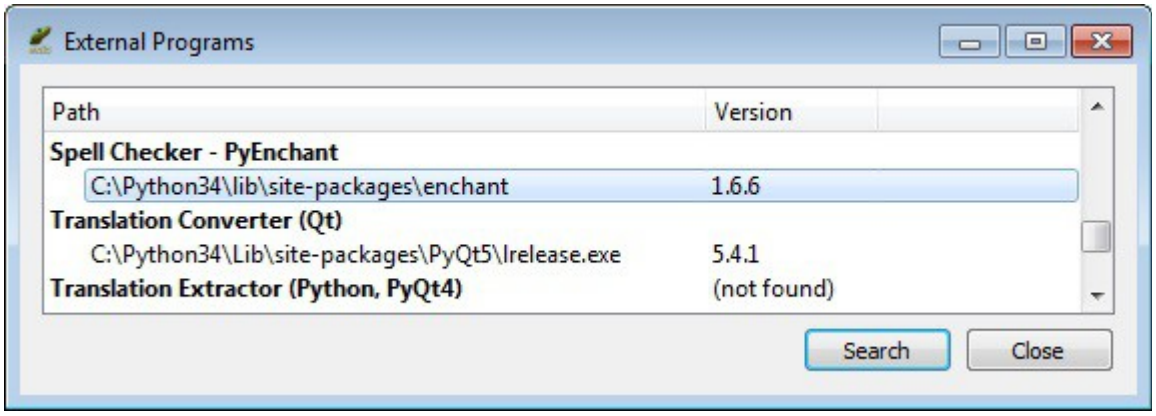
<。> Just entered a plain setup-run, we got a "`pyenchant-1.6.6-py2.7.egg-info`" into the "`C:\Python3X\Lib\site-packages`" directory, and a "`Python 3.4 pyenchant-1.6.6`" into the Control Panel list; and there also reassuringly well `Uninstall`-able.

– –

## *Then*

Once PyEnchant installed, at next Eric's run you'll have it listed on the Eric's "`External Programs`" list [*verify via:* `Settings > Show External Tools`],

154[*ver. 1.6.6, latest in Oct. 2015*]




and also have all the `Extras > Check Spelling...` commands turned enabled, with all PyEnchant's spell-checking features smoothly available [*verify*].

– –

### *"Mutatis mutandis"*

This is to remind the reader that the following sections will constitute a valid technical and practical reference only provided that in each actual operative situation all what should be changed—such as actual URLs, or revision codes—be considered, and assumed as adequately updated and changed by a conscious reader. That is, saying it concisely in Latin: "*Mutatis mutandis*"

– –

## *Keeping Pace*

As said [*ref. above sec.:* Version Selection Sequence], current Report's edition [*S-PM 160400*] is about Eric ver. 6.1.*n*, considered as compatible with Python 3.4.3 and PyQt 5.5.1. Now, taking into account that major changes are already announced both for PyQt (→5.6), Python (→3.6) and Eric (→6.2), how long will this Report probably last before becoming painfully obsolete? One year, at most; not more.

Then, of course, not all of this Report will have such a short life span, only some sections here and there. Anyhow that will be enough to undermine its overall practical usefulness, to instill uncertainty to the reader and also uneasiness to the Author, conscious, alas, of not being anymore in condition of keeping such a frantic working-pace. Well, now, is there anything that can be done? Perhaps: <。> *Candidates proposing positive solutions are welcome…*

-<>-

# Setting of Required Packages

*S-PM 160400*

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

– –

## *1/2]* Python (3) – *ver. 3.4.3*

Programming language Python 3, the main toolkit here needed first [*cf.:* Required Packages *above*].

### *Where*

Available for downloading from URL: `http://www.python.org/download/releases/3.4.3/`

### *What*

Download file: `•Windows x86 MSI Installer (3.4.3) (sig)`

That is, among the various items there available, the latest Python 3 version compatible with current Eric, Windows x86, MSI Installer – binary only, no source included[155].

<。> Note that here we are not yet considering the latest Python 3 version available in absolute[156], but the latest currently compatible with the subsequent prerequisite [*see next sec.s*].

### *How*

Setup-run, with `Administrator` permission. All defaults accepted, and you'll get such a: "`C:\Python3X`" root directory installed [*verify*].

All items on the resulting Windows system `Start > Python` button are of immediate comprehension, but perhaps the "`Docs Server`" item, aimed at running "`pydoc`", a tool to manage Python modules documentation.

### *Then*

Then a test-`Start` run of the "`IDLE (Python GUI)`" program, to verify that it's working all right.

### *Uninstall*

Under the resulting system `Start > Python` button there is also a reassuring "`Uninstall Python`" command [*nicely working, as we had chance to verify*].

– –

155With the cryptic "`(sig)`" as short for "*PGP signature*", to possibly verify the integrity of the downloaded file.

156That is, at present: 3.5, and announced: 3.6;

<。> By the way, comparing it with the preceding ver. 3.4, you'll see that the type of installer has changed, from "`.msi`" to "`.exe`": that's precisely what we meant to say beforehand with the warning-section "***Mutatis Mutandis***" [*see*].

## *How to Recover*

### *a badly corrupted installation*

During an assumedly usual test activity we happened to get a Python installation so badly corrupted[157] that not only its `Repair` procedure wasn't an efficacious remedy, but even the `Uninstall` couldn't work. Then, remembering that Python, for the most part, was totally self-contained in its `\<Python3X>` root installation directory, we dared an action that revealed successful, as hereafter described [*just in case...*].

In case the Python `Repair` and even the `Uninstall` procedures reveal ineffective, try:

- Delete the entire `\<Python3X>` installation directory;
- this way the Python `Repair` procedure should turn effective; then just run it.
- Thus also the `Uninstall`, and a possibly subsequent re-`Install` or upgrade procedure, would work. At least it did in our case.-

– –

## *2/2]* **PyQt (5)** – *ver.* 5.5.1

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.

– –

## *Viewpoint*

*<~> This is a tool hosted in a rather variegated web site where you'll find both source material, related with the very PyQt development, and also ready-to-use material.*

*Fortunately then things are much simpler than one might expect, as you simply need to go to the "`Binary Packages`" section where, at least with the Windows platform, the ready-to-use package includes also all other related tools listed by Eric as distinct pre-requisites[158]. Which they are not, in the sense that you don't need to tell the difference and to look for each one of them[159], as they are already included into this very same PyQt package.*

– –

157Err.Msg.: "Requested Python version <3> is not installed", "Python is not a valid Win32 application. Access is denied."

158[*cf. URL:* http://eric-ide.python-projects.org/eric-download.html]

159That is, to look for Qt (from Digia) and PyQt (from Riverbank), as distinct items; plus QScintilla.

<~> A not-so-clear situation that we anyhow accept "AsIs", without committing into any further investigation.

## *Where*

PyQt can be found at URL[160]:
`http://www.riverbankcomputing.com/software/pyqt/download5`

## *What*

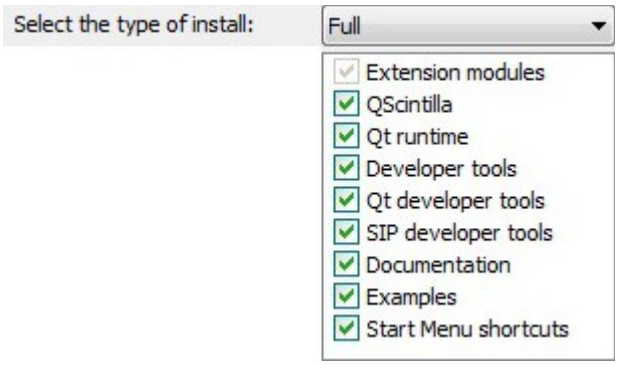
What you here need [*see:* Essentials, *table* Version Reference, *in:* {0.Lead}] is simply the Windows installer: `PyQt5-5.5.1-gpl-Py3.4-Qt5.5.1-x32.exe`
[*most recent, Eric compatible — cf. above sec.:* Required Packages][161]

As for: GPL - General Public License, Python 3.*x*, Qt 5.*x*, 32 bit. Just a few minutes of downloading.

– –

## *How*

A setup-run, with `Administrator` permission, all defaults accepted, and you'll get a: "`\PyQt5`" into the very Python directory—of course here assumed already installed—under the "`\Lib\site-packages`" [*see*]. This the "`Full`" list of the installed items:




## *Then*

After the installation, on the Windows system `Start` menu you'll see such an item:

`PyQt GPL v5.5.1 for Python v3.4 (x32)`

160And there so defined: "PyQt is the Python bindings for Digia's Qt cross-platform application development framework." Added: "Note that the Qt documentation is not included.", with a referred link to: http://qt-project.org/doc/; that is to signify a distinction between the very Qt documentation and the PyQt's, here well available.
161That is, ver. 5.5.1 – The current announced as under development is ver.: 5.6

That's a folder rather rich with programs, `Documentation`, `Examples`, ... . So rich that, without a proper "primer" documentation[162], it's easy to get lost.

- -

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

```
>>> from PyQt5 import QtCore, QtGui
```

If smoothly accepted, it shows that you're up and running.

- -

## *Uninstall*

Under the resulting system `Start > PyQt` button there is also a reassuring a "`Uninstall PyQt`" command. We've prudentially executed it from here before each possible successive upgrading.

-<>-

162Which we haven't been able to find out, so far. Alas.

# Eric (6)

*S-PM 160400*

Here it's about the Eric toolkit, final and chief subject of all this setup sequence. More precisely it's about Eric ver. 6, as an IDE (Integrated Development Environment) for Python ver. 3.

## *Where*

Eric (6) can be found at the URL:
`http://sourceforge.net/projects/eric-ide/files/eric6/stable/`
conveniently pointed at by a link into the very Eric's web site, `Download` section.

## *What*

Just download: `eric6-6.1.n.zip` [*most recent*]

## *Remark*

Other Eric items there available for downloading, such as:
`eric6-<x>.zip`
`eric6-<x>.tar.gz`
…
`eric6-i18n-de-<x>.zip`

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;<br>otherwise, without such marker: default language (English) |

<!> In this Report we are concerned exclusively with Eric intended as an executable program, not as a developing project. Therefore here we'll deliberately ignore all about the wealth of source material comprised into the downloaded Eric package[163]. A package here regarded as a functional "black-box", and exclusively from a user's point of view.

-<>-

163Anyhow at least a look inside this package, and inside its sub-directory `\eric`, is highly recommended [*see*].

### *How*

Just expand the "`.zip`" package, then locate and run the "`install.py`" script procedure there inside[164]. This way you'll get, into the very Python installation directory, a sub-directory "`\eric6`" added into the "`\Lib\site-packages`", and there also a rich set of such "`eric6.bat`" commands into the `\Scripts` sub-directory [*verify*].

### *Remark*

This same Eric (6) "`install.py`" script procedure is meant to be executed by the version of Python currently installed, be it ver. 3 or 2. That means that, in case you've already got installed into the same computer both these versions—a not so infrequent condition—to install and then run Eric with the desired interpreter you can simply specify it explicitly[165], this way:

```
C:\Python3X\Python.exe install.py
```

Thus Eric, as a program, will be then interpreted and executed exclusively by that Python version used to install it. Whereas then Eric, as an IDE, will possibly run and use either Python ver.s, at user's choice [*cf. property:* `Progr. Language`, *at command:* `Project > Properties...`].

<~> In case, for instance for test purposes, you want to install and use *both* Eric flavors executable by Python ver.s 2 and 3, you may run the "`install.py`" script with the "`-y`" switch, so to add the so called Python variant "`_pyN`" to the program's name; e.g.: `eric6_py2` or `eric6_py3` [*cf. on-line help:* `>C:\Python3X\Python install.py -help`, *or hereafter sec.:* Eric's "`install`" Options].

– –

### *Viewpoint*

<~> Another and, to some extent, more elegant and stable way for overcoming the (minor) difficulty for such a choice, would simply be to make a good use of the dedicated source file extensions "`*.py2`" and "`*.py3`", in place of the generic "`*.py`".

In fact, as we've successfully done in our workstations, you just need to properly specify, say: `install.py3` and `uninstall.py3` where needed, and then inform the Windows operating system which is the Python interpreter to be associated with the desired "`pyN`" extension (e.g.: `C:\Python3X\python.exe` ).

### *Viewpoint*

<!> But, to this purpose, our deeply rooted opinion is quite another. Such a problem simply shouldn't exist at all, as a fairly conceived professional tool, such as Python aspires to be, *should evolve without breaking compatibility with just the previous version number*.

---

164A ">…`\Python install.py -h`" is to display some related help info [*see*].

165Other possible actions to grant Eric the desired selection of one Python ver. over the other, is to install first the desired one, then the other, after the Eric's installation. Or to manually modify the related `Path` environment variable.

With time, obsolete statements could be abandoned, or deprecated; that's well acceptable. But not turned into incompatible, that's unwise and unfair. Otherwise, instead of a progressive evolution, what you get is the current Python 3 vs. Python 2 antagonism, and mess. Indeed.

– –

Now, this a typical Eric installation log:


Where you're also reassured about "`All dependencies ok`". That is, in this case:

*1/4]* `Python 3.4.3`
*2/4]* `Qt 5.5.1`
*3/4]* `PyQt 5.5.1`
*4/4]* `QScintilla 2.9.1`

A list showing in particular that the toolkit `PyQt`, being already installed as a prerequisite [*see sec.: 2/2]* PyQt (5)], brings along also the toolkits `Qt` and `QScintilla`, which, clearly, here don't need to be handled individually.

Then, as first step in the sequence that follows, you have a "`Cleaning up old installation …`", implying that this same procedure can well be used for updating Eric over a preceding installation and, in that case, with all "historical" configuration and preferences kept [*cf.: next* Uninstall *sec.*].

<。> Then, after "`Installation complete`", you'll have that nothing new will appear on the Windows system `Start` button, so that it is advisable to add there a custom shortcut for running Eric, as described in the next "Custom Eric `Start` Command" section [*see*]; that is, unless you'd rather keep running Eric as described in the next "*Then*" section.

– –

# *Then*

At "`install.py`" completion, a test run of the "`eric6.bat`" command made available into the `<Python3X>\Scripts` directory could be used to verify that everything is working properly [*verify*].

## *Remark*

All the other similar "`*.bat`" commands, such as "`eric6-api.bat`", …, which can be spotted into the same directory along with "`eric6.bat`", are meant to run various Eric tools autonomously. For instance "`eric6-webbrowser.bat`" is to run the Eric Web Browser independently from Eric [*verify*].

– –

## *Uninstall*

Into the same original package, besides "`install.py`", you'll find also a reassuring "`uninstall.py`" procedure [*see, and see also the above* Remark *about running such procedure in case of both Python 3 and 2 installed*]. No message at execution completion[166].

Note that, as said, in case of an Eric upgrading, the `install.py` procedure can be called directly—that is: without a preventive `uninstall.py`—as its preliminary action will be anyhow a "`Cleaning up old installation...`" [*cf.: installation log snapshot, here above*].

<。> But note also that there is a difference between that preventive upgrade–uninstall and this explicit call. Indeed, in this latter case, you'll have the choice whether to keep or not your current working conditions; being asked:

```
Found these plug-in directories
  - C:\Python34\Lib\site-packages\eric6plugins
  - C:\Users\User101\_eric6\eric6plugins
Shall these directories be removed (y/N)? …
```

166<~> Frankly here we'd appreciate such a final "`Uninstall done ok`" message.

```
Found the eric data directory
  - C:\Users\User101\_eric6
Shall this directory be removed (y/N)? …

Found the eric configuration directory
  - C:/Users/User101/AppData/Roaming/Eric6
Shall this directory be removed (y/N)? …
```

– –

### *Viewpoint*

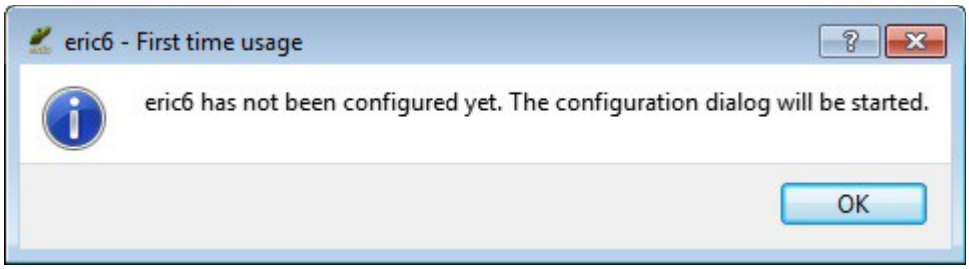
Here we'd rather welcome the possibility of making a fair difference between a sheer-uninstall, that is when you really want to "get rid" of an installation, and an "upgrading"–uninstall; that is when the keeping track of such elements as configuration, history parameters and preferences is required, and welcome.

Indeed it's our opinion that a fair uninstall process, when required, should be capable of erasing *every aspect* of the related stuff, leaving the host system precisely as it was before the corresponding installation, clearly and neatly. That is to say, just for one thing, Windows System Registry included. <~> All of us know how degraded and jammed an operating system can become, after a successive series of not-so-clean uninstall procedures. Do we not?

– –

## Eric "`First time usage`"

The Eric "`First time usage`" will be intercepted for any new user, so to set up a new operative configuration.



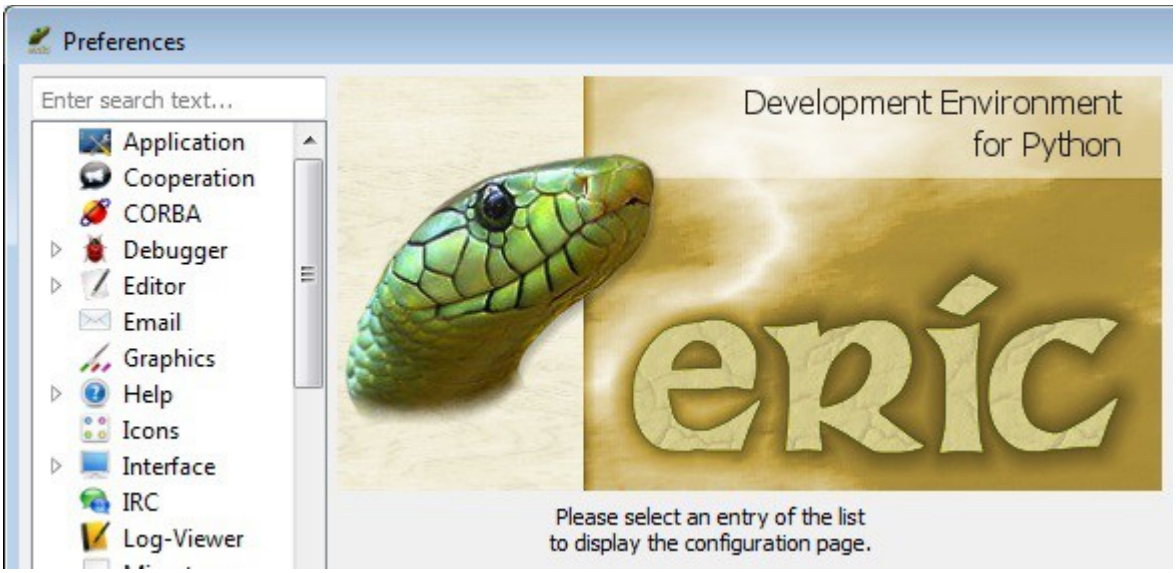

### *Viewpoint*

<~> As we already knew, at a set-up completion it is not so infrequent to be confronted with a modal window where you'll be forced to configure some of the working parameters you'll enjoy during this new phase of your professional life; disregarding the fact that this might precisely be the moment when you possibly know less about this new condition of yours. It's like asking a just new-born baby his preference about the wall-paper for his room.

Luckily most of the times you'll be permitted to escape such an initial ordeal, and postpone it to a better time. Alas it was not the case with Eric, therefore, intolerant to arbitrary impositions as we are, we resolved to break somehow the execution — with a consequent error condition at the next Eric run. Never mind, freedom has a price.

– –

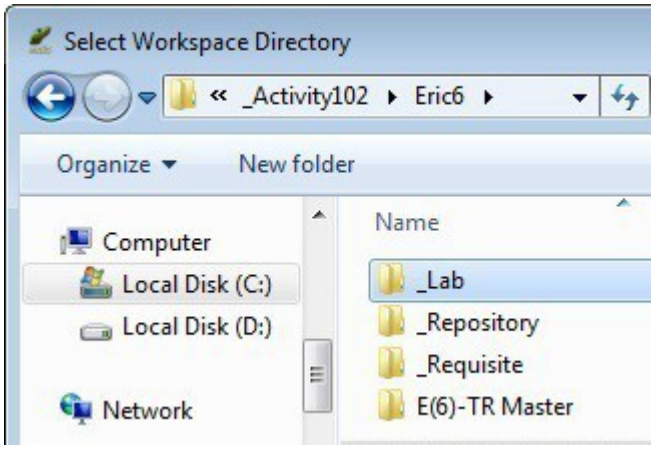
After that, you will be then requested to open a configuration page, and possibly there set your preferences. Specific Eric preferences will be treated all over this Report just there where needed [*e.g.:* `Settings > Preferences... – Application, Configure the application`]; anyhow some suggestions of general interest have been here offered at Eric Setup Completion section [*see hereafter*].




In any case, this same condition can be brought up running independently the `eric6_configure.bat` macro command that you'll find in the `<Python3X>\Scripts` directory, or also at any other moment during a usual Eric session, with command: `Settings > Preferences...`

– –

After that, you'll be then invited to select an initial default value for the Eric Workspace directory, which will be the default root-directory where to host future Eric Projects' material [*cf.: command* `Project > New...`].




Anyhow it's a default value then changeable at any moment, on the `Workspace` field of: `Settings > Preferences... - Project > Multiproject, Configure multiproject settings` [*see*].

– –

On subsequent runs on the same station, and by the same user, even after a new Eric setup, you'll be not asked again, as all such configuration data (as, just for example, the "`\_Lab`" directory shown in the above snapshot) will be kept, unless you'll perform the following *Reset User Configuration* procedure.

– –

### *Reset User Configuration*

If desired, Eric "`First time usage`" condition can be reset simply erasing the "`*.ini`" files you'll fine inside[167] your `<User>\AppData\Roaming\Eric6` "hidden" directory[168]; next Eric run will thus execute as it were the first.-

– –

## Eric's "`install`" Options

So far, in this chapter, we have found convenient to consider Eric's `install.py` as a neat procedure capable of doing its job without requiring any particular parameter, or data. But that is not completely true, in the sense that, more precisely, it is a command-line so neatly defaulted not to require any particular parameter, or data, *most of the times*.

167Three of them, at most: `eric6.ini`, `eric6recent.ini`, `eric6messagefilters.ini` [*about this last see:* `Settings > Edit Message Filters...`].

168As a rule "`\AppData`" is a "hidden" directory, so that, to see it, you have got first to set its related "`View`" option to "`Show hidden files, folders`".

That's why, in this section, we'll offer complete description of Eric's `install.py` syntax, precisely for those few times when some special option might come badly useful.

– –

But first, two words about the conditions under which an "optioned" `install` execution can be run, as they certainly are not so obvious.

- Ascertain that the `install`'s file extension "`.py`" is associated with the `Python (Console)` program; via system command: `Open with > Choose default program...`
- Ascertain that the `C:\Python3`*X* installation directory is referenced into the "`Path`" Environment Variable — required to launch the Python program without having to specify its full path; via system: `Start > Control Panel`, `System > Advanced system settings`
- Operate within a `Command Prompt` (DOS) shell, and there move to the directory hosting the Eric setup stuff — required in case you'd rather call the `install` file without having to specify its full path.

– –

The above conditions fulfilled, and assuming `C:\`*`myDir`* as the directory hosting the `install.py` file, any such command line will become executable:

```
C:\myDir>install --help
```

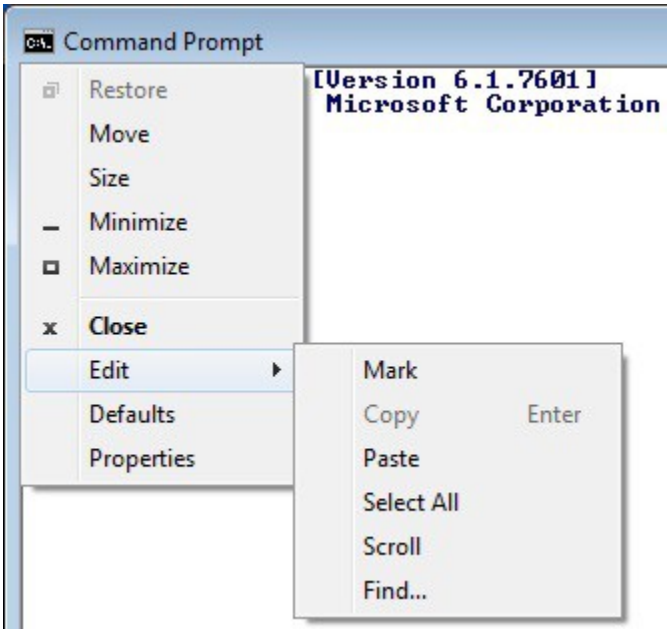
The above "optioned" command, for example, is to print the related online documentation.

– –

## *Remark*

Having to operate on a DOS shell you'll have to renounce all comforts offered by the Windows graphical interface, and also to a friendly communication between these two environments (that is: DOS – Windows).

But, at least for a shared copy/paste of texts, there is a remedy: it's the `Edit` tool on the pop-up menu associated to the Title Bar icon of the Command Prompt shell [*verify*].




This `Edit` tool will at least permit you to copy/paste textual data, such as long path–strings and online documentation texts, and possibly interchange them between these two otherwise separated environments (through the common system Clipboard).

– –

### *Online Documentation*

Here is how the Eric `install` procedure will describe itself, when run with the switch: `--help` (short form: `-h`)

Note that such execution, but for this printout, is purely dummy, and nothing will be installed.

```
C:\myDirX>install --help

Usage:
    install.py [-chxyz] [-a dir] [-b dir] [-d dir] [-f file] [--pyqt=version]
where:
    -h, --help display this help message
    -a dir     where the API files will be installed
               (default: c:\Python34\Lib\site-packages\PyQt5\qsci\api)
    --noapis   don't install API files
    -b dir     where the binaries will be installed
               (default: c:\Python34)
    -d dir     where eric6 python files will be installed
               (default: c:\Python34\Lib\site-packages)
```

```
    -f file    configuration file naming the various installation paths
    -c         don't cleanup old installation first
    -x         don't perform dependency checks (use on your own risk)
    -y         add the Python variant to the executable names
    -z         don't compile the installed python files
    --pyqt=version version of PyQt to be used (one of 4 or 5)
                   (default: 5)

The file given to the -f option must be valid Python code defining a
dictionary called 'cfg' with the keys 'ericDir', 'ericPixDir', 'ericIconDir',
'ericDTDDir', 'ericCSSDir', 'ericStylesDir', 'ericDocDir', 'ericExamplesDir',
'ericTranslationsDir', 'ericTemplatesDir', 'ericCodeTemplatesDir',
'ericOthersDir','bindir', 'mdir' and 'apidir.
These define the directories for the installation of the various parts of eric6.

Press enter to continue...
```

– –

### *Options, Commented*

In this section we'll examine more in detail each one of the switches and options above listed.

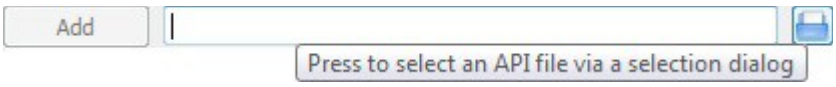
`-h, --help` `display this help message`

`-a dir` `where the API files will be installed`
`(default: C:\Python3X\Lib\site-packages\PyQt5\qsci\api)`

*Comment*

It's to assign a custom location to the standard API info file set, as it will be used by command `Edit > Complete` [*see*]. Useful in case the given default location, for any reason, is not available.

Note that then, when required




[*see:* `Settings > Preferences... – Editor > APIs, Configure API files`], it is up to the user to select them (API), as Eric will not be able to do it automatically.

`--noapis` `don't install API files`

*Comment*

This API info file set is an Eric standard resource, not a pre-requisite, therefore you are free NOT to possibly have it.

`-b dir` `where the binaries will be installed`
`(default: C:\Python3X)`

*Comment*

Once installed, Eric is to be run through such "`eric6.bat`" command, default-located into the Python directory [*verify*].

This option[169] is provided so to possibly locate it elsewhere, along with all other of its companions (`eric6_api.bat`, ..., `eric6_webbrowser.bat`). A useful possibility when, for any reason, you'd rather use a custom location.

```
-d dir          where eric6 python files will be installed
                (default: C:\Python3X\Lib\site-packages)
```

*Comment*

An option similar to the above one [*see*], aimed at the entire bulk of the Eric stuff. Note that in this Report, for obvious practical reasons, we'll refer exclusively to standard default locations[170].

```
-f file         configuration file naming the various installation paths
The file given to the -f option must be valid Python code defining a
dictionary called 'cfg' with the keys 'ericDir', 'ericPixDir', 'ericIconDir',
'ericDTDDir', 'ericCSSDir', 'ericStylesDir', 'ericDocDir', 'ericExamplesDir',
'ericTranslationsDir', 'ericTemplatesDir', 'ericCodeTemplatesDir',
'ericOthersDir','bindir', 'mdir' and 'apidir.
These define the directories for the installation of the various parts of eric6.
```

*Comment*

A feature based upon the preventive production of a configuration script, possibly useful for the production of distribution packages—typically: Linux's—rather then for a single installation action. As an example see "`eric6config.py`", in the Eric's standard package.

The parameters here considered—`'ericDir'`, ..., `'apidir'`—are first defaulted in the standard "`install.py`" procedure, with some of them shown in this very `--help`, as `(default: …)`

```
-c              don't cleanup old installation first
```

*Comment*

The "cleanup" action is equivalent to a preventive execution of the "`uninstall.py`" procedure [*see also, here above:* ***Viewpoint****, at Eric's* Uninstall *sec.*]. This switch is to prevent such action, with a consequent installation process implying a simple copy of new files, with overwriting of the possibly old ones.

```
-x              don't perform dependency checks (use on your own risk)
```

*Comment*

It's to force anyhow the installation process without the preventive check of the Eric's prerequisites [*see sec.:* Prerequisites*, here above*]. You should have a good reason for doing so.

---

169 Would you wander why this option refers to "binaries", even though no binary files are here involved, it's in obedience to the tradition, as so are usually referred the executable "`.exe`" program files.

170 <~> Needless to say that any deviation from standard default conditions may bring about a certain amount of risk...

`-y` `add the Python variant to the executable names`

*Comment*

This switch is to add the so called Python variant "`_pyN`" to the program's base name (extension "`*.pyw`" or "`*.py`" unaltered); e.g.: `eric6_py2` or `eric6_py3`, instead of simply: `eric6` [*cf. in dir.:* `C:\Python3X\Lib\site-packages\eric6`].

This way, if required, you may install and distinguish into the same station both `eric6` program editions, as interpreted by Python ver. 2 and 3 [*see above: related* Remark*, at the* Eric (6) *installation sec.*].

`-z` `don't compile the installed python files`

*Comment*

A switch to prevent the generation and, therefore, the use of the so called Eric's "`*.pyc`" *Compiled Python Files* or, more appropriately, *bytecode* Python files. Bytecode or *p-code* (portable code) Python files execute more efficiently than the corresponding "`*.py`" source code, are aimed at an abstract machine and are not human-readable.

`--pyqt=version of PyQt to be used (one of 4 or 5)`
`(default: 5)`

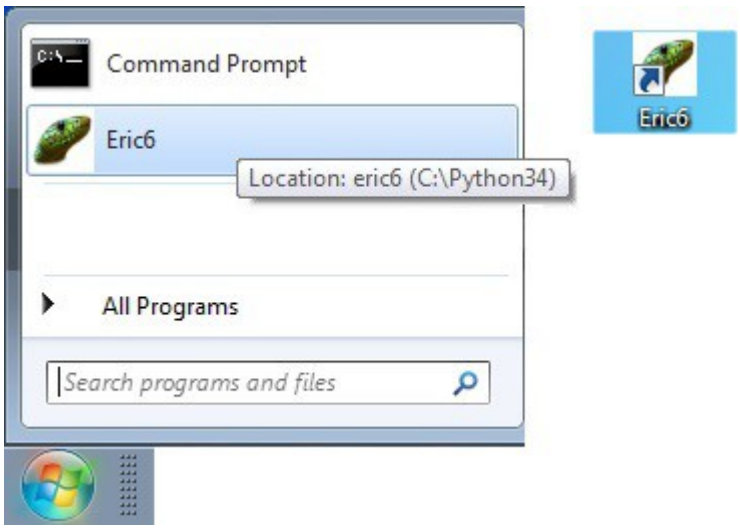
*Comment*

Eric is fully compatible with both current PyQt's versions, 4 or 5. When both available, the default preference is for the most recent, with this switch possibly taking precedence.

-<>-

## Custom Eric `Start` Command

<!> For running Eric it is convenient to create, under the Windows system `Start` button, a shortcut referring to the standard "`eric6.bat`" command[171] as it results available into the `<Python3X>\Scripts` directory [*verify*]. Here is how to create such a useful shortcut:




*1]* Locate the `eric6.bat` command into the `C:\Python3X\Scripts` directory;

*2]* Right-click it to "`Create Shortcut`", and keep this shortcut there;

*2.1]* Into the `...\site-packages\eric6\pixmaps` directory there is a nice `eric6.ico` that you could assign to it;

*3]* A drag-drop copy of such shortcut can be added into the system `Start` menu, possibly renaming it as "`Eric6`", or also on the desktop [*see figure above*]; keeping anyhow also the original shortcut as with the former point.

– –

### *Remark*

No instruction is here offered about how to run *separately*—although well possible—each single application handled by Eric, mainly Python. That's because Eric, being an IDE, is already designed to run automatically anyone of them, whenever required, without any special user intervention.

-<>-

171A look at the text of this "`.bat`" command and you'll see that it refers to an "`eric6.pyw`" script, whose "`.pyw`" extension implies the execution of program `Pythonw.exe`, not `Python.exe` [*about this matter see standard* Python Manuals*, section:* 3. Using Python on Windows].

# Eric Setup Completion

The Eric operative environment obtained with an initial successful installation is rich enough to deserve—even require—some subsequent actions of setup completion, all to be performed via menu command: `Settings > Preferences...`, as hereafter described.

## *Remark*

It may be worth knowing that Eric saves all working parameters into a dedicated "`eric6.ini`" file, distinct for each log-on user. In case you want to give it a look, here is where to find it:

…\<*User*>\AppData\Roaming\Eric6

Note that, as a rule, the `\AppData` is a "hidden" directory, so that, to see it, you have got first to set its related "`View`" option to "`Show hidden files, folders`". Then you'll be able to spot it and verify that, say, such setting as the "`Single Application Mode`"—as just hereafter described [*see next sec.*]—, is recorded into this "`.ini`" file of yours, `[UI]` section, "`SingleApplicationMode`" key.

– –

## *Menu command:* `Settings > Preferences...`

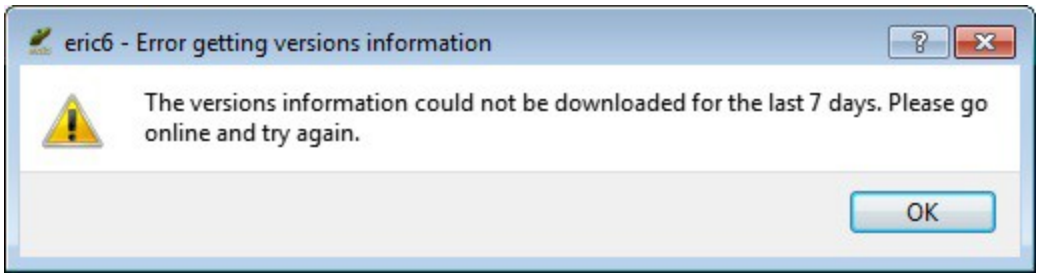
<。> *On control form:* `Application, Configure the application`

`Single Application Mode`

Check-box provided to possibly run only one single instance of the Eric application, at a time. When checked, a possible second call of Eric will supersede, not add to the current one.

`Check for updates`

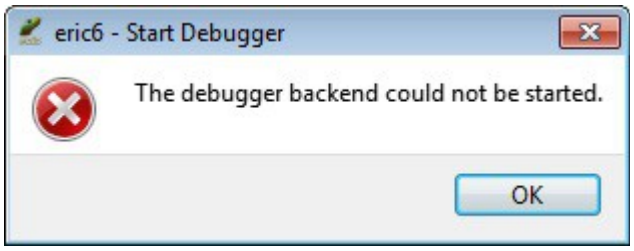
Set of option buttons where to possibly select the option "`None`", so to avoid such a unexpected "`Error`" condition at some Eric run:




– –

<。> *On control form:* `Debugger > Python[x], Configure Python[x] Debugger Python[x] Interpreter for Debug Client`

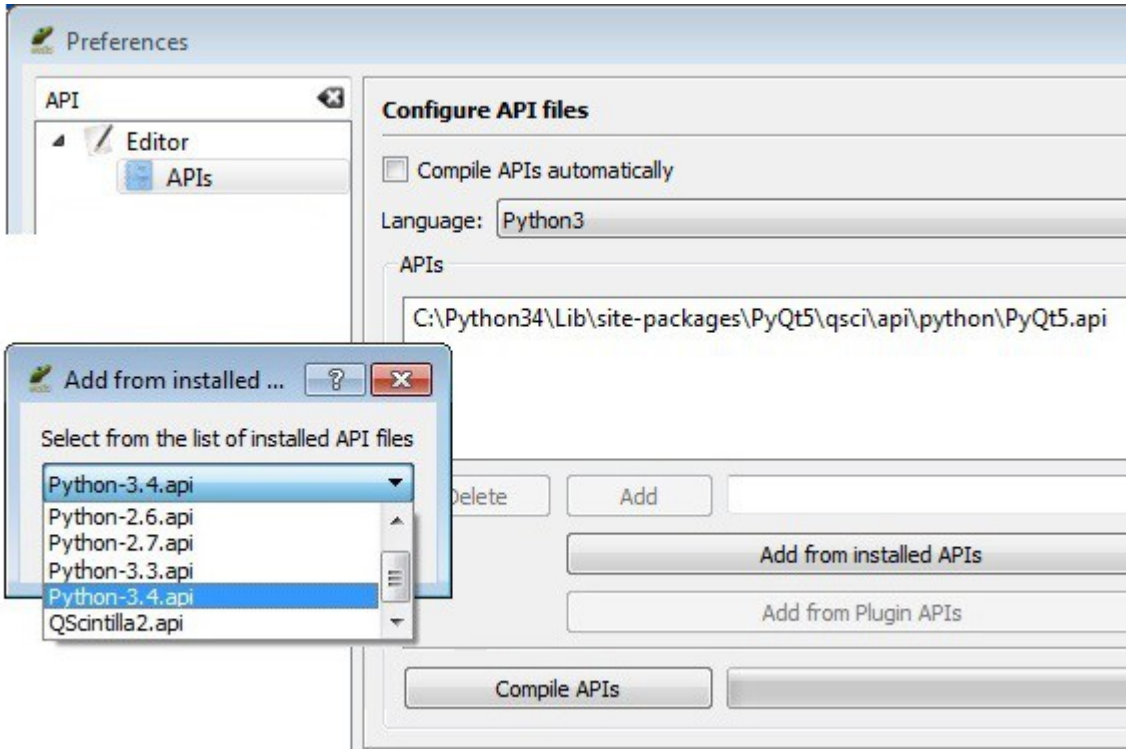
This is the area to inspect, and "fix", should such an error box appear:




– –

<。> *On control form:* `Editor > APIs`

It's where to select and compile (i.e.: enable) the so called API instruction files, as required by the Autocomplete and Calltip features [*see command:* `Edit > Complete`].



`Language` To select the target programming language.
<~> You'd better take note of it, as there is no way to later know which it was, among the many here available.

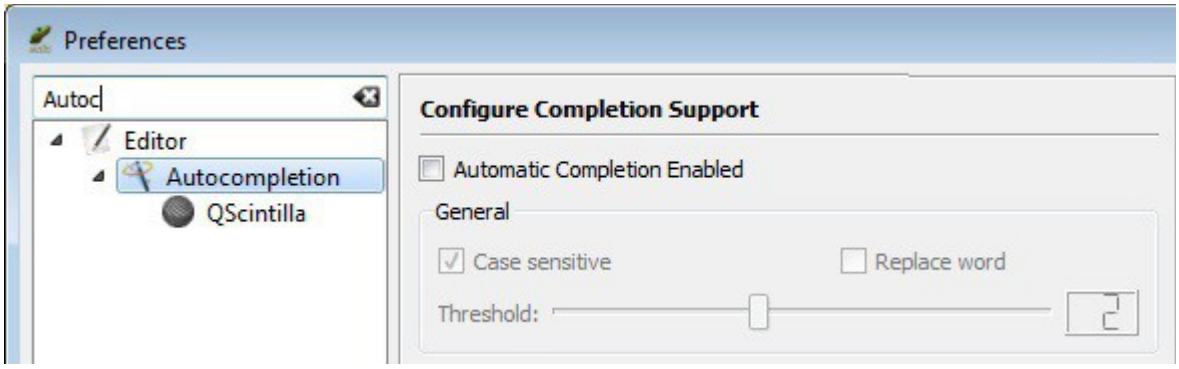
`Add from installed APIs`
That is, those available after a standard Eric set-up (no others have we here considered).

`Compile APIs`
Required to enable the added items (actually, it's a binary conversion, for speed reasons).
No need here to select any single API info file, as it will act on all the listed ones. Manual action correspondent to "`Compile API automatically`" [*verify*], being it disabled or, for any reason, ineffective.

– –

<。> *On control form:* `Editor > Autocompletion`

Here is where to configure the Autocompletion feature and possibly switch it from "manual", that is activated via `Edit > Complete` commands [*see*], to "automatic", that is activated automatically on-the-fly, as you type the source text.

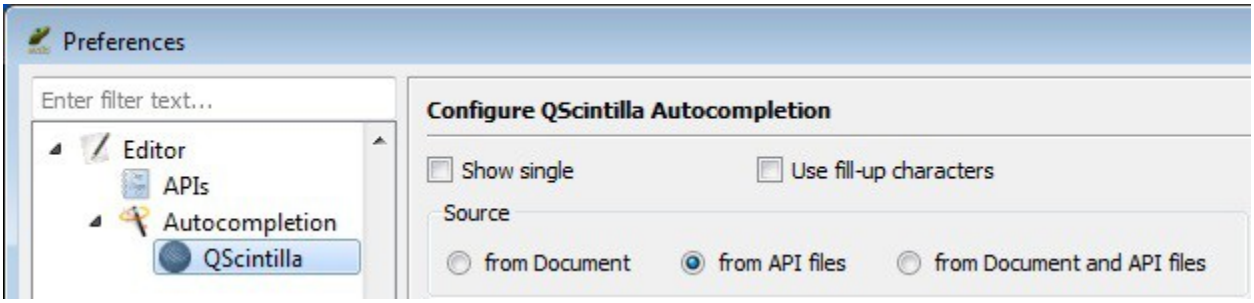



`Automatic Completion Enabled`
Check-box to switch the operating mode from manual to automatic, with the feature's status that remains anyhow enabled.

– –

<。> *On control form:* `Editor > Autocompletion > QScintilla`

Then here is where to manage the QScintilla Autocompletion tool[172], as available upon a standard Eric set-up.

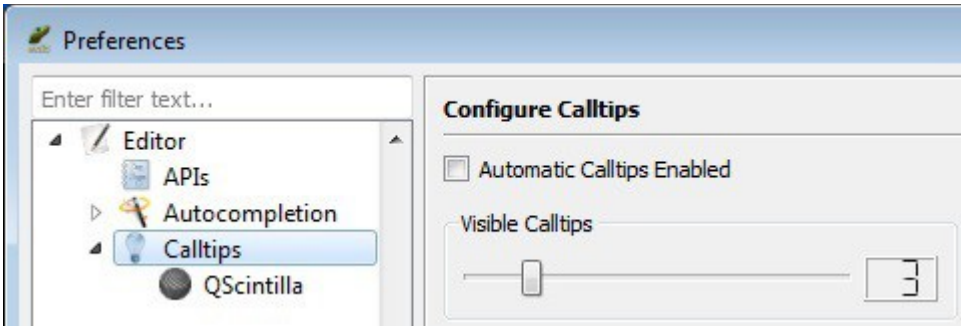



Particularly relevant the `Source` choice [*see*].

– –

<。> *On control form:* `Editor > Calltips`

And here is where to manage the Calltip feature, all self-explanatory enough not to require any further explanation.




Here too, as with the Autocompletion feature [*cf. above sec.*], the "`Automatic Calltips Enabled`" check-box is for choosing between a manual/automatic operating mode, not between an enabled/disabled status.

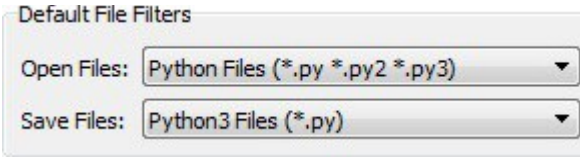
– –

172cf. the above "`All dependencies ok`" snapshot and list, at the Eric set-up section.

<。> *On control form:* `Editor > Filehandling, Configure file handling settings`

`Default File Filters`

Drop-down lists, as for instance:

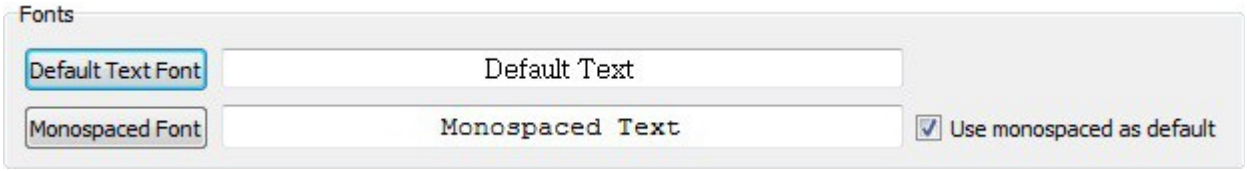



Where to conveniently change the default file extensions in case the current values were not of your taste.

– –

<。> *On control form:* `Editor > Style, Configure editor styles`

`Fonts` configuration area, with also a: `Use monospaced as default` check-box




## *Remark*

Font configurations that may appear as ineffective if you are not aware that:

- This font selection will refer to *future* source files, not to the possibly current ones;
- "`Monospaced Font`" is simply a label, not a condition, as in this field *any* available font can be selected and set; that is proportionally spaced too [*cf. context command:* `Text Form (^) Use Monospaced Font`].

– –

<。> *On control form:* `Help > Eric6 Web Browser, Configure web browser`

`On startup:` `Show Home Page` (instead of: `Show Speed Dial`)

So not to forcibly require a preventive Internet connection active at each "`Help > Helpviewer...`" execution [*see*].

– –

<。> *On control form:* `Help > Help Documentation, Configure help documentation` `Python` and `PyQt5 Documentation`[173] — Text fields where to set such paths as:

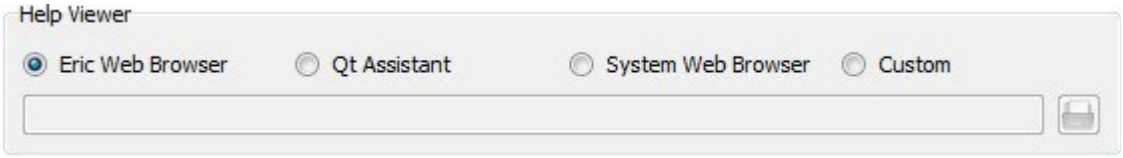
```
C:\Python3X\Doc\python343.chm
C:\Python3X\Lib\site-packages\PyQt5\doc\html\index.html
```

## *Remark*

According to what declared in this `Help Documentation` form, these same parameters could be entered as such Windows environment variables as: PYTHON3DOCDIR [*verify*]; manageable via DOS "`set`" command[174], or Python "`os.environ`" dictionary.

– –

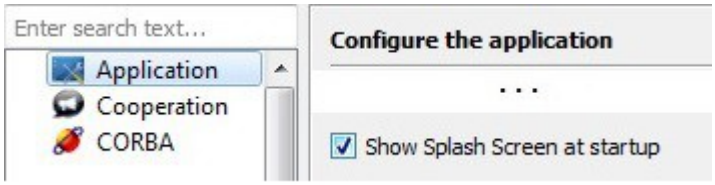
<。> *On control form:* `Help > Help Viewers, Configure help viewers` `Help Viewer`




A set of option buttons, where to possibly select: `Eric Web Browser`

– –

<。> *On control form:* `Application, Configure the application, Show Splash Screen at startup`




Check-box where to enable/disable the very appearance of the Eric splash-screen.

## *Remark*

Would you like to gain full control over the appearance of the Eric splash-screen `ericSplash.png` and such other images as `eric_small.png` and `eric.png`, this is the directory where to find them: `C:\Python3X\Lib\site-packages\eric6\pixmaps`

There just modifying a file name to, say: `-eric_small.png`, you could "hide" it altogether from Eric; with no error condition generated [*as we've verified*].

– –

173<~> Sorry for not having here any suggestions to offer for Qt5.
174<~> We have to confess that, after a first hasty and unsuccessful attempt, we've renounced exploring this possibility.

## Eric's *Local* and *Global* Scope

As seen in the above "Eric Setup Completion" section, and as you'll realize throughout all your working experience, many are the configuration parameters characterizing the Eric's functionality, conveniently described where needed throughout this Report, specifying also their storage location and operative scope [*e.g.: "Local" (Project's) / "Global" (User's) Dictionaries, by command* `Extras > Edit Dictionary`].

Anyhow there is an Eric specific distinction among *Local* and *Global* scope and storage locations[175] which is convenient to consider here, in general:

*Local* scope — intended at *per-Project* level, as in: `<Project>\_eric6project\...`, for such items as: `*.e4d`, `*.e4p`, `*.e4s`, `*.e4t` files [*verify*]. That is, operating conditions equally valid for any User working on a given Project. Notably:
`Project > Properties...`, as in `*.e4p` file [*see*].

*Global* scope — intended at *per-User* level, as in: `<User>\_eric6\...`, for such items as: `/Downloads` sub-dir, `eric6session.e5s` file [*verify*]. That is, operating condition equally valid for any Project a given User is working on. Notably:
`Settings > Preferences...`, as in `<User>\AppData\Roaming\Eric6` [*see*].

- -

Eric Plugins' special Global install location [*see:* `Plugins > Install Plugins...`]:
`<Python3X>\Lib\site-packages\eric6\Plugins\...`, visible to all Python's users.-

-<>-

175<~> That is, standard for Eric, as it's hard to find two distinct products assigning the same meaning to these same terms. For instance, with Python variables, *Local* and *Global* terms assume rather different role and meaning [*cf.:* `Window > Debug-Viewer`].

# Some Optional Packages

Here is a list of some optional packages that can be installed at wish, at any moment, in no special order. What here follows are some real-case, offered as notable examples.

`Python ver. 2` Can be installed beside the pre-requisite Python ver. 3, as here considered in this Report, so to actually enjoy what declared in section "Compatibility with Python ver. 3 or/and 2" [*see in:* {0.Lead}]. For technical details refer to the former "*1/2]* Python 3" section, changing what is to be changed[176].

`PyQt4-(Py2)` PyQt companion GUI library for Python 2 [*cf. above sec.:* *2/2]* PyQt (5)].

`cx_Freeze` A popular cross platform set of scripts and modules for "freezing" Python scripts into executable programs, distributed under the PSF open-source license. A subject here treated as a notable example for command: `Plugins > Install Plugins...` [*see in:* {1.North}].

`Mercurial` and `Subversion`
Two well known Version Control Systems (VCS)[177], here considered with the menu command `Project > Version Control` [*see in:* {1.North}].

`PyEnchant` Spell-checker toolkit, here considered as a notable example with command `Settings > Show External Tools` [*see in:* {1.North}], required by menu command `Extras > Check Spelling...`, and also by "Check Spell Menu" context menu [*see in:* {2.Central}].

– –

## *Viewpoint*

<。> That of the Eric optional packages and add-ons is a not-so-little world of its own, with some related info that can be found here at the command `Plugins > Install Plugins...` and, more in general, on the very Eric web site [*see*].

It's certainly a topic out of the main-stream as here assumed, but nevertheless about which we'd welcome Mr. Reader's specific expressions of interest and experience, in view of possible future editions of this Report, or of other dedicated booklets.

– = –

176Or the other Report about Eric 4, entirely based upon Python 2, as available at the Eric web site [*see:* Foreword, *What & Where in the Internet*].

177<。> About PySvn Subversion, for Eric 4, and Mercurial, for Eric 5, a dedicated Report has been produced, well valid for Eric 6 too, and available in the Eric web site as with the preceding footnote. Mercurial, in particular, is assumed to be the main VCS for Eric.

# Appendix

## Sections and Revision

This whole Report is composed of distinct "*Sections*", delimited by such Begin-Of-Section (BOS):

*S-PM 160400*

and such End-Of-Section (EOS) lines:

– –

These other EOS codes, with increased visual evidence: "–<>–" (End Of Page) and "– = –" (En Of Chapter), may be used instead of the ordinary "– –".

<!> The "*yymmdd"* code on the BOS plays the role of *date-revision code*:

Each Section can be edited independently, with the most recent date that implicitly supersedes older ones. Sections possibly still unprovided with such date-revision code are supposed to inherit that one of the first BOS put on the cover page of the Report.

This explicit revision mechanism[178] is assumed to play a role on the future life of this Report.

– –

### *Chapter Prefix Code*

For all Chapters—that is: main and numbered Sections—it has been introduced such a "{*n.Tag*}" prefix code, reproduced on all page headers as a convenient reference and navigation tool:

| | |
|---|---|
| {0.Lead} | Intro |
| {1.North} | North Side, on the Eric Application Window |
| {2.Central} | Central Park |
| {3.South} | South Side |
| {4.West} | West Side |
| {5.East} | East Side |
| {6.Trail} | Setup & Management |
| {Map} | Map of the Eric Application Window |

–<>–

178Besides the usual disclaimer telling that: "The information in this document is subject to change without notice" [*see:* Copyright Page].

# Typographical Conventions

Prevalent typographical conventions here adopted to increase readability.

| | |
|---|---|
| Arial | This the font reserved for titles. |
| Capitalized | Usual text font, with the initial capitalized not only for proper names, such as: Eric or Python, but also to qualify terms to be intended in a specific technical sense, such as: Working Copy, Tag, Branch. |
| `Courier New` | For standard technical elements and references, such as:<br>URL `http://sourceforge.net/projects/`<br>File path `C:\Program Files\`<br>Menu command `Settings > Preferences...` |

## *Viewpoint*

To be extra-precise such "`Settings > Preferences...`", when intended as an Eric command, should be typed with font `Courier New`, whereas, when intended as the title of the related Report's section, should be typed as "Settings > Preferences...", that is with font Arial.

<~> But, frankly, our love for formal accuracy doesn't go so far, and we'll generally go in favor of `Courier New`, leaving to you the burden of interpreting it strictly as a command, or as the section describing it, or as both.

– –

| | |
|---|---|
| *`Italics`* | Segment within a `Courier New` string, referring to a replaceable/variable item, such as a generic *`filename.ext`* into a standard path: `C:\Dir\Subdir\`*`fileneme.ext`* |
| `>` | Separator for menu command path, e.g.: `Extras > User Tools > Select Tool Group` |
| `(^)` | "click" action symbol, for context pop-up menu, e.g.: `Context (^) Show > Code Coverage...` |
| <.> | Mark for a point deserving special attention — As:<br><!> Remarkable point<br><?> Questionable point<br><~> Perplexity, doubt, suggestion<br><...> Ellipsis, or Topic still to be completed<br><.?.> Questionable point, still under investigation |
|  | Icon for an [*Eric Feature Under Revision*], and whose Tech.Report is currently left AsIs. |

– –

| | | |
|---|---|---|
| Abbreviations: | cf. | *confer* – compare with |
| | cmd | command |
| | e.g. | *exempli gratia* – for example |
| | i.e. | *id est* – which is to say |
| | ref. | refer to |
| | sec. | section |

– –

# Reference Book List

*S-PM 160400*

This Report is not intended as, and cannot conceivably be, a substitute for the technical reference of the very products integrated into Eric. That is, mainly: Python language and Qt graphic library. On the contrary, it is the user's knowledge and active interest for such products that will act as a conditioning prerequisite for an effective Eric's usage.

Therefore we assume that it would be convenient if, close-by this Report[179], you kept handy such good reference books as those hereafter listed.

Python v2, v3 Documentation
: Python Software Foundation, as with the Eric `Help` command [*see in:* {1.North}]

Python Essential Reference
: by David M. Beazley, Addison-Wesley

PyQt 5 Reference Guide
: Riverbank Computing Ltd, as with the Eric `Help` command [*see in:* {1.North}]

Rapid GUI Programming with Python and Qt
: by Mark Summerfield, Qtrac Ltd

– = –

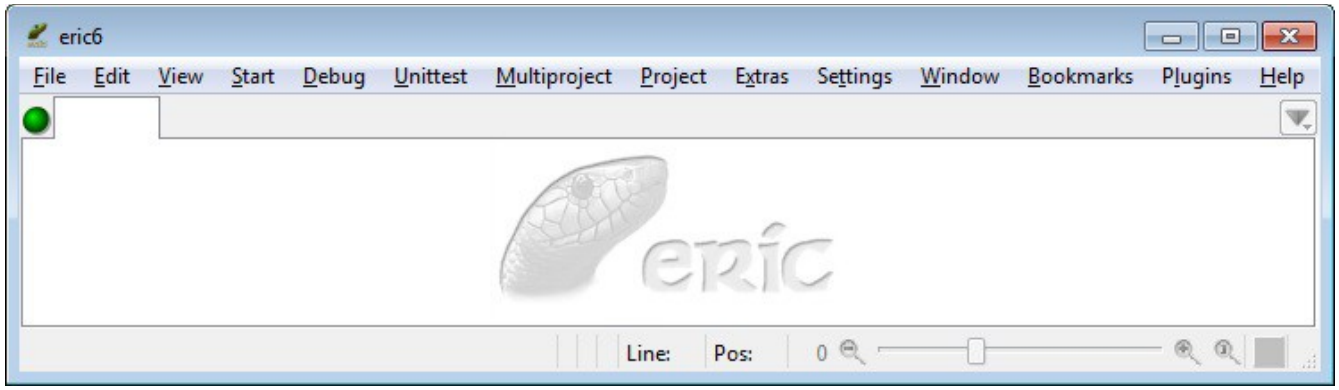
179And, why not, also the other Reports as available on the Eric web site [*see:* Foreword, *What & Where in the Internet*].

# Table of Contents

– = –

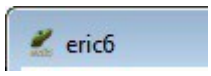
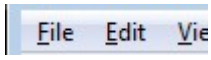
# {Map} Eric Window Map
## and Glossary

Maps of the Eric (6) application window, aimed at locating all of its graphic objects, listed along with their standard denomination and a reference to the section of this Report where they are specifically treated.



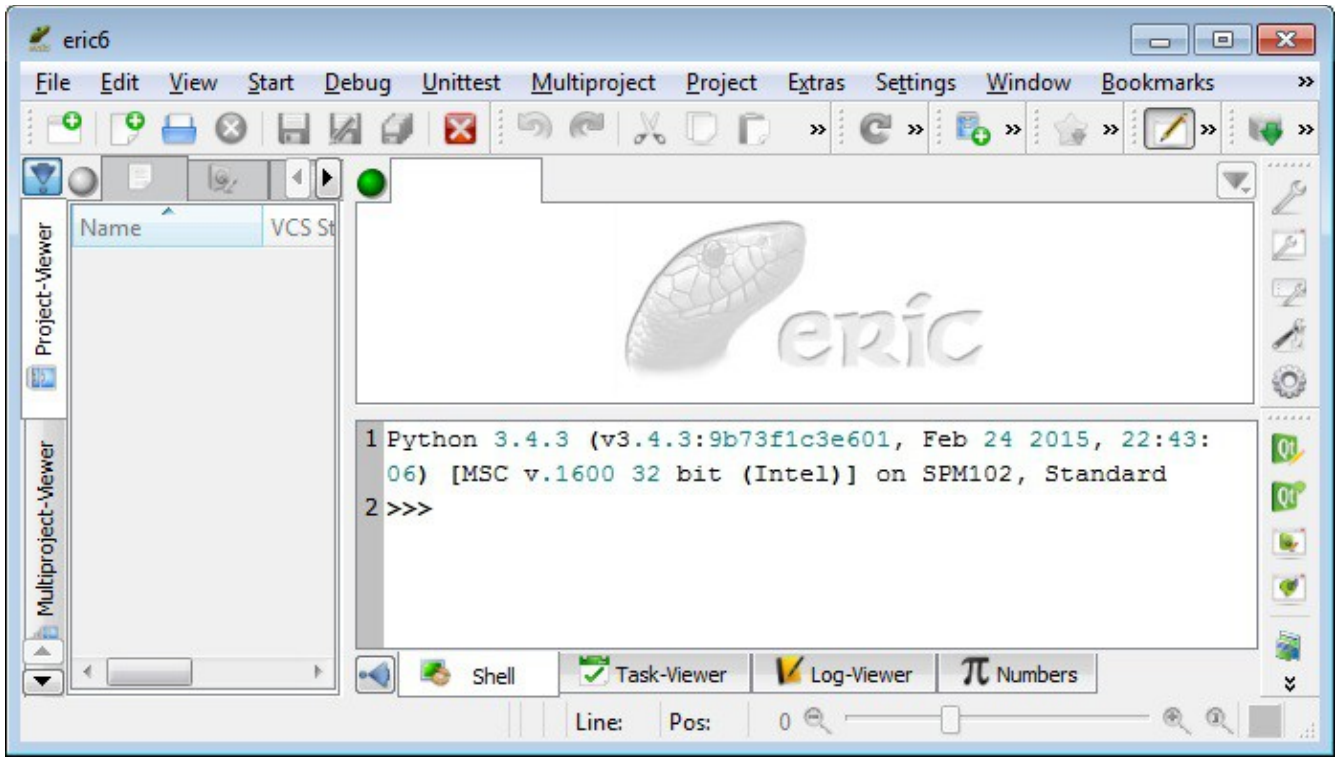

This the Eric Window in its simplest appearance, corresponding to the condition as set by the main menu command `Window > Edit Profile`, with all related items—that is: Sidebars and Toolbars—un-checked [*verify*]. Configured this way, the work area hosts exclusively one form, the *Central*, that is the Tagged Text Form, where Python source modules are edited. None of the Auxiliary Forms—Left, Bottom, Right—nor the Toolbars are here shown [*cf. next map*].

– –

| *Graphic Item* | *Denomination* | *Ref. Section* |
|---|---|---|
|  | Title Bar | {1.North} |
|  | Menu Bar | {1.North} |
|  | Text Edit Form | {2.Central} |
|  | Information Bar | {3.South} |

–<>–

Here is the usual Eric Window, in its much more crowded initial default appearance, where to locate all the main graphic objects not shown on the preceding, simpler, map [*cf.*]. That is:



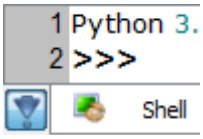

Auxiliary *Left*, *Bottom*, and *Right Pane*[180], located around the stable *Central Pane* on the application work area; plus the horizontal and vertical *Tool Bars*.

– –

*Graphic Item* *Denomination* *Ref. Section*

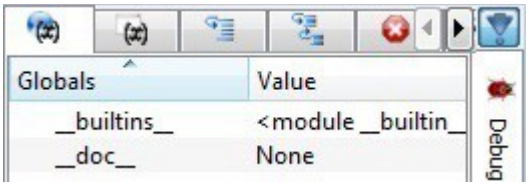
Tool Bars ..... {1.North}

180Actually added hereafter, apart.




Bottom Pane ..... {3.South}


Left Pane ..... {4.West}




Right Pane ..... {5.East}

-<>-

# Glossary

*S-PM 160400*

Glossary of terms as used throughout this Report, referring to items on this Eric application window, listed in inclusion order.

*Window* Top graphic operating structure of an application on the *Desktop*, that is the computer screen.

*Bar* Window frame stripe, for such elements as: Title, Icon-Tools, Menu.

*Pane* Main, adjustable subdivision of the work area within a Window, to host specific Controls and Forms.

*Control* Generic name for such elements as: button, command, selection check-box, option round-box, list-box, …

*Dialog Box*, or simply: *Box*
A secondary Window for specific user interactions. Likewise, just a bit more specific: *Control*, *Error*, *Tool*, *Alarm*, *Hint Box*.

*Form* Homogeneous area of a Window, sub-Window or Pane, organized for a specific task; typically with controls and labeled fields to be filled with textual data.

*Tab* A control for selecting one among possibly different overlapping Forms. In a Tab, if the case, you may distinguish *Tab-label* and *Tab-page* areas.

*Source Text Form*
Or Text Edit Form, or simply Text Form: the Main Form of this application, where source text is edited[181]. With these typical contents, expressed in Python terms:

*Script* Single or main source Python file;

*Module* Source Python file meant to be imported into a main Script. Anyhow this is a term not so infrequently used interchangeably with Script.

A main Script with its Modules can be collectively referred to as a Program. Anyhow it is here worth recalling that Project is a specific Eric concept, with no precise correspondence in Python terms.-

– = –

181So that, coherently: a "*Text Editor*" is the tool you use to write into the "*Text Form*".