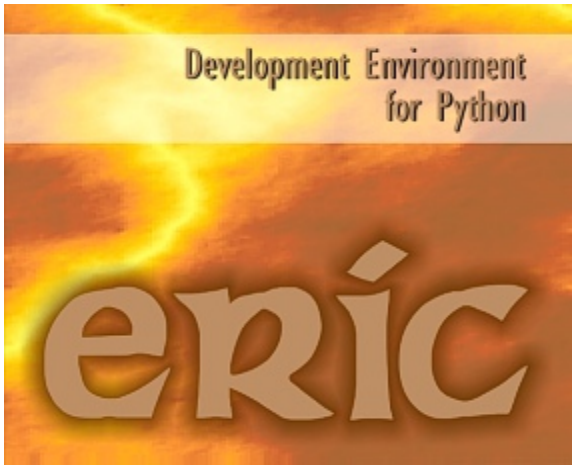


S-PM 130900

Eric

Python 3 (& 2)
Integrated Development Environment

Technical Report

Copyright Page

S-PM 130700

No commercial uses, No modifications allowed; Jurisdiction International, Format Text
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

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.)

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

– = –

Foreword

S-PM 130500

Dear Reader,

*Here is this "*Eric (5) – Technical Report*" to enrich the collection of the* Reports *already produced about Eric 4 [*see*], under the same basic principles characterizing this whole work.*
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.*

See you.
The Author

– –

What & Where in Internet

Eric Discussion	Mailing List:	eric@riverbankcomputing.com
Eric Site	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

– = –

{0Lead} *Essentials*

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

S-PM 130700

Version Reference

Nr	*Item*	*Version*	*Note*
1	Windows XP Pro Windows Vista Ultimate	`5.1 (SP 3)` `6.0 (6002:SP 2)`	Host Operating System
2	`\Python33`	`03.03.02`	Host Directory
3	Eric	`5.3.5` `(rev 2736ebd219bb)`	IDE Application

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

Prerequisite Python 3 required (plus Python 2, admitted), with PyQt graphic library
[*see:* Prerequisites, *in:* {6Trail}]

Execution Command: `\eric5.bat` (into directory: `C:\Python3x`)

- -

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 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[1], among which are the Eric subjects to be considered off-scope for this Report, and therefore here just overlooked. Namely the Special Features and the Special Details , as hereafter listed [*see*].

-<>-

1 Then, of course, there certainly are also some un-intended, unintentional limits. Surely another story.

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.

Examples:

Version Control System[2]
: 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*];

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

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

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

Translations
: As with such tool as the Translations Viewer, that is: "a tool to load and display Qt User-Interface and translation files for a selected language", and the Qt Linguist [*see command:* `Extras > Tools (Select Tool Group = Builtin Tools)> Translations Previewer...`];

Web Browser[3]
: That is: "a combined help and HTML file browser" [*see command:* `Help > Helpviewer...`];

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 available to the Eric users community [*see section:* *R-Pane*: IRC*, in:* {5East}];

Project Packaging[4]
: That is: "Generate a distribution package using cx_Freeze" [*see command:* `Project > Packagers > Use cxfreeze`];

---

2 Actually, about this topic an "Eric 4 – Version Control Technical Report" has been produced, substantially valid for this Eric (5) too [*see:* Foreword*, What & Where in Internet*].

3 Actually, about this topic an "Eric 4 – Web Browser Technical Report" has been produced, substantially valid for this Eric (5) too [*see:* Foreword*, What & Where in Internet*].

4 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 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.

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

Debug Remote, Configurable
As with commands: `Settings > Preferences... – Debugger` and `Project > Debugger` [*see*];

PEP 8 Compliance, Syntax and Tabnanny Checks
As with command: `Project > Check` [*see*];

– –

Project Deployment: a Neglected Feature

<!> Looking at such an impressive list of ***Special Features***[5], some of which, as a 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 in Eric, but for a modest front-end to call an external "`cxfreeze`" script, available as a sub-command of the `Project > Packagers` set [*see*].

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 care of the operating system, but interpreted care of the very Python interpreter, assumed as ubiquitously present in all computers. Of course we know that this scheme is not always valid, and that things are not so simple. Indeed we've produced a dedicated Report precisely to examine to some detail[6] what here has been just hinted at.

– –

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

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

Special Details

Lots of Eric minor 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:

Similar or identical commands,
: such as: `Open...`, `Close`, `Save`, available in different menus, which will be described only once, then simply referred to [*e.g. commands:* `Multiproject > Open...` *and* `Project > Open...`*, as referred to:* `File > Open...`].

"`Configure Email`" parameters,
: such as the `Outgoing mail server (SMTP)` and the `Outgoing mail server port`, as requested for command `Help > Report Bug...` and `Request Feature...` [*see*].

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:* {1North}].

-<>-

True Multi-Platform

Eric is a true multi-platform product, and as ubiquitous as Python. That is, in principle, 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.

– –

Python 2 – 3 Compatibility

The concurrent presence, and use, of the languages Python 2 and Python 3 is, assumedly, the reason of the equally concurrent presence of Eric 4 and 5 IDEs. And, as a consequence, also the reason why, after the "Eric 4 Report" [*see*], here we are engaged with this other analogous Report, centered on Eric 5. 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 successive generations of the same product[7], in a process that looks more like a secession than a transition, as it seems to be with Python 2 and 3. And, as a consequence, also with Eric 4 and 5.

Initially we had no other choice but to realize and accept things as they were, with Eric 4 to begin with. But now we have happily realized that Eric 5, beyond Python 3 required as a prerequisite, can equally use and manage also Python 2. Plus we have been informed that while Eric 4, now on, will be simply maintained, Eric 5 will be actively updated and developed. Therefore, at least in prospective, we assume it is envisioned to become the unique IDE for all current Python flavors.

And this is a prospective that we're glad to adopt here right away, starting with the title of this Report, where no "5" appears. As to say that we already envision this Eric (5) simply as *the* Eric Python IDE, *tout court*.

Remark

<!> Note that Eric, besides being multi-platform, is also somewhat a multi-language IDE, as it can handle both Python 2, Python 3 and, marginally, also Ruby[8] [*see next section*].

– –

7 More than four years, now. A geologic era, with informatics.

8 Ruby support is comparatively rather limited, in particular for the debugger.

Migration Python 2 → 3

A consequence of the here stated Python 2 – 3 compatibility is that Eric (5) 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*].

Viewpoint

<!> That is, a developing environment where to possibly alleviate the perverse consequences of the incompatible evolution from Python 2 to Python 3.

– –

Python 2 – 3 (and Ruby), Easy Selection

Note that besides Python 3, which is an Eric (5) prerequisite, *also Python 2 can be installed* into the same computer system, along with its specific PyQt4 library version [*see:* Prerequisites, *in:* {6Trail}]. With such a set-up configuration, Eric can offer the choice about which the Python version to use with any given project, as hereafter described.

Remark

<!> Within this Eric IDE also Ruby modules[9], with "`*.rb`" file extension, could be interleaved, and also with the debugging feature maintained[10]. 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:* {2Central}].

– –

How to Select the Language Interpreter

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

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

– –

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

10 Anyhow a feature that we're not intentioned here to explore, as assumed off scope for this Report.

- *File Extension* – For each single Python module, by means of the module 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[11]:

  `#! [...] 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.

Remark

Note that, in itself, the Shebang line has a role of setting the path to the interpreter only with Unix and Unix-like platforms, is ignored by Windows and has a role with Eric only for the Language Interpreter selection as here described, in all platforms, Window s included.

– –

- *Editor flag* – For each single Python module, inserting an Eric specific "Editor flag"[12] 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.

– –

Then, according to such Python 2 – 3 choice, also these other editing aids, as the "Typing Completer" and the "Language Highlighter", will be automatically selected, as for the setting on: `Settings > Preferences... - Editor, Highlighter, Filetype associations` [*see*][13].

Remark

<!> Note that, this way, within the same Eric Project, both Python 2 and Python 3 modules can be freely intermixed[14] [*see also:* `Project > Properties... - Mixed programming languages`]. This also as a possible way for a smooth transition from Python 2 to Python 3.

– –

11 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`. Names not recognized are simply ignored.

12 No other "*`Keyword = Value`*" pair so far defined for this *Editor flag* mechanism.

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

14 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.

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 pop-up context menu command: `Shell (^) Start`, on the Bottom Form [*see in:* {3South}].

– –

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 encoding-comment:

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

  as first or second line on a Python module[15]. A standard Python convention valid for Eric too.

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

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

S-PM 130500

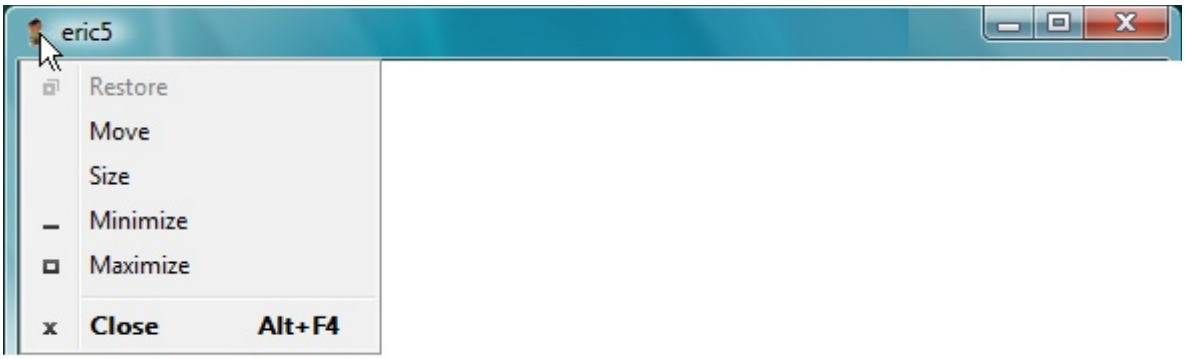
Looking at the Eric Application Window as it were a city-map we can distinguish five main areas: a *North* and *South Side*, a *Central Park* and an *East* and *West Side* [*see also:* Eric Window Map*, in:* {Map}*, at the end of this* Report].

This {1North} section is about the North Side, where are located the: *Title Bar*, *Main Menu Bar* with all its commands, and various *Tool Bars*.

– –

Title Bar

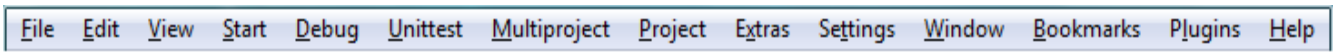
"Eric5" Title Bar, positioned, as usual, on top of the application window, with standard icon menu and Minimize, Maximize, Close buttons.



-<>-

Main Menu Bar, and Commands

Menu Bar, on its standard position just below the Title Bar, where drop-down Command Menus can be called. It's the main subject of this section.



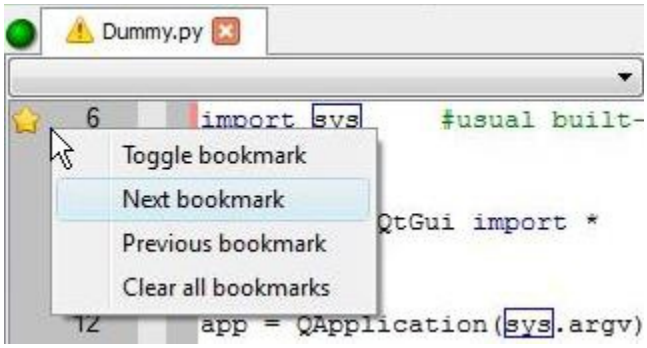
Menu Index

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

– –

Remark

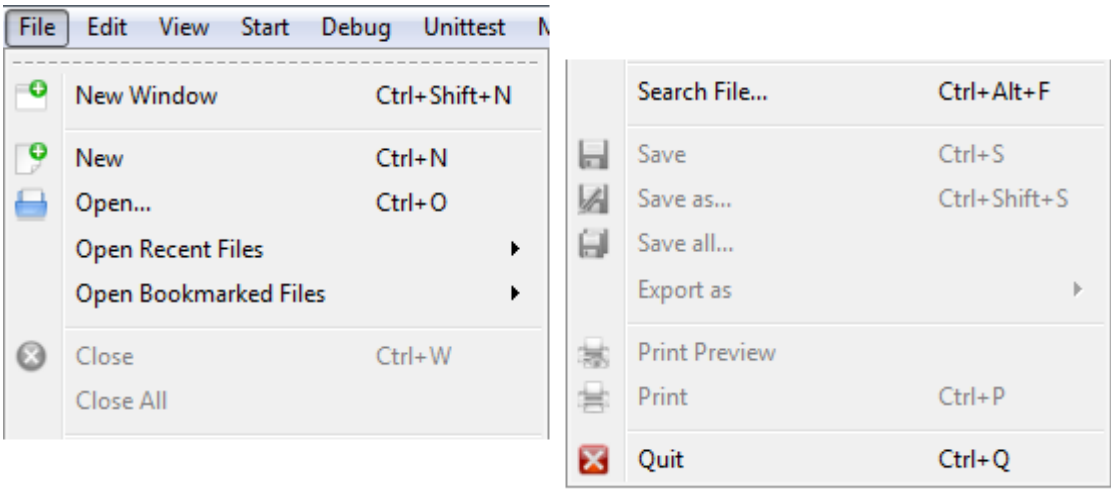
<!> However much important this main menu might be, it doesn't exhaust all the operative power of Eric, which indeed lays in great measure on the many, many pop-up context menus, associated to the various controls of the Eric application window. Pop-up context menus as shown in this example,



conveniently callable with a mouse right-"click"—here symbolically represented with a "(^)"—and as hereafter described in full detail in 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 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

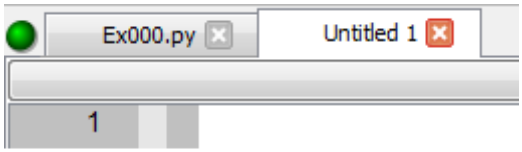
That is, provided the application is not configured as "`Single Application Mode`" [*see command:* `Settings > Preferences... – Application, Configure the application`].

– –

File > **New**

Designed to create a new *Tagged Text Form* into the Eric *Text Pane*, where to edit source text [*see:*

Application Window Map, *in:* {Map}]; with default name: "`Untitled n`".



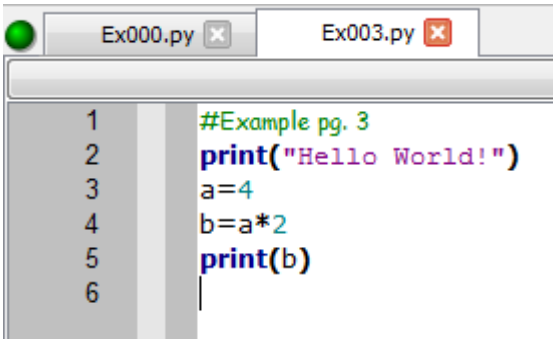
Remark

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[16] a "`*.py`" Python source file, on a new tagged text edit form, named after the selected file.



This command is exclusively aimed at opening single files, not Projects or Multiprojects, which have got their own dedicated `Open` commands [*see*].

Remark

<!> Both Python ver. 2 or ver. 3 source files (`*.py2` or `*.py3`) can be here opened and properly interpreted, as explained at section Python 2 – 3 Compatibility [*see in:* {0Lead}].

Initial default file type as with command `Settings > Preferences... - Editor > Filehandling`,

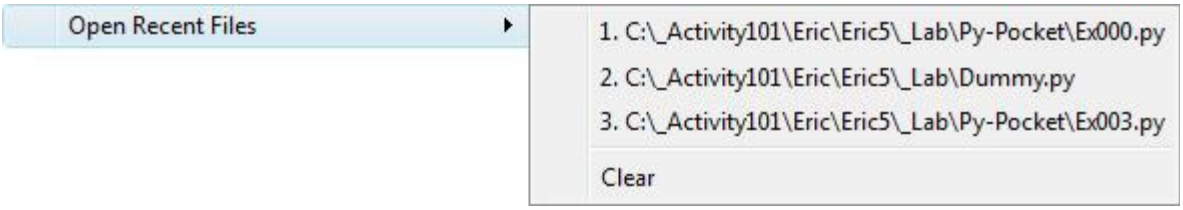
16 Typically but not exclusively, as any text files could be here opened.

`Default File Filters, Open Files`. Which should be manually preset to such a convenient "`*.py`" value, as the original is an uncomfortable "`*.bat`", simply for alphabetic reasons. [*see also:* Eric Setup Completion, *in:* {6Trail}]. Then the default directory and file extension are as with 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 [*see command:* `File > Open...`].



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

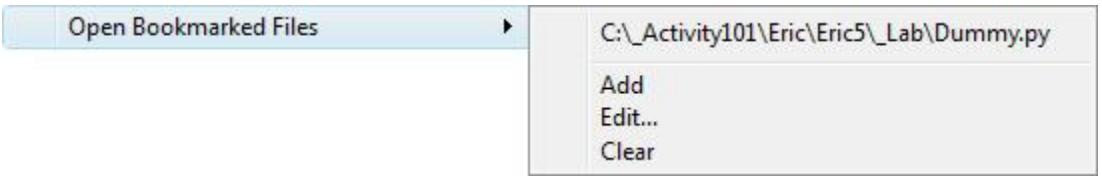
Remark

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

– –

File > Open Bookmarked Files

Designed to show the current list of "Bookmarked Files" 标记的文件



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

`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[17];

`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

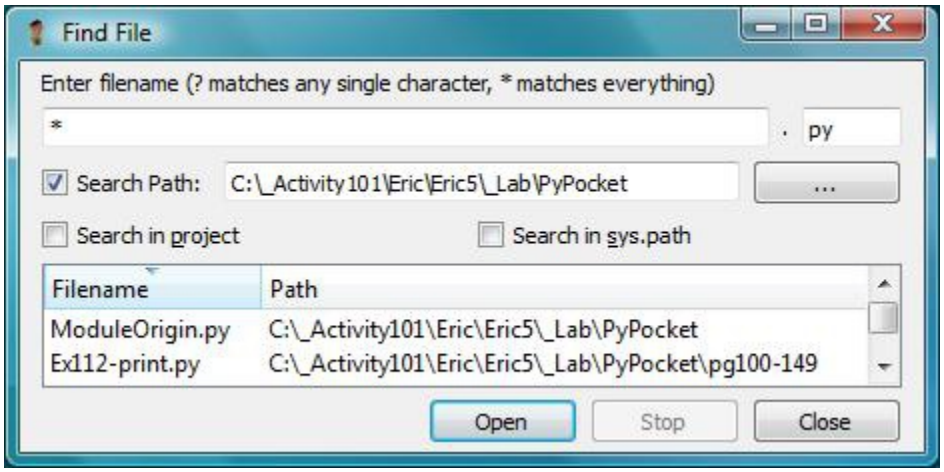
In case of modules belonging to an opened Eric project [*see command:* `Project > Open`], the project as such remains open.

– –

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.

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



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 the contents of the currently active tagged Text Form into the related original file, which 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 Python

modules when opened with the `File > Open` commands, but also when opened with the equivalent `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 the specific data strictly related 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 former paragraph implies that Eric makes a distinction between the sheer Eric Project and the set of source modules comprising the same Project[18].

– –

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.

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

– –

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. The original source remains unaltered.

This the available format choice[19]:

18 And if such a subtle distinction comes out a bit baffling to you, be reassured: you are not alone.
19 A set of formats with just a historical origin.

`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

– –

Viewpoint

<~><~> No sensible default value is here offered for the destination file and directory, only for the file type, alas. While we'd consider as natural to be here offered, as default, the same name and directory of the original file.

– –

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)`].

Remark

Parameter "`Magnification`", in: `Settings > Preferences > Printer – Configure Printer Settings`, 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.

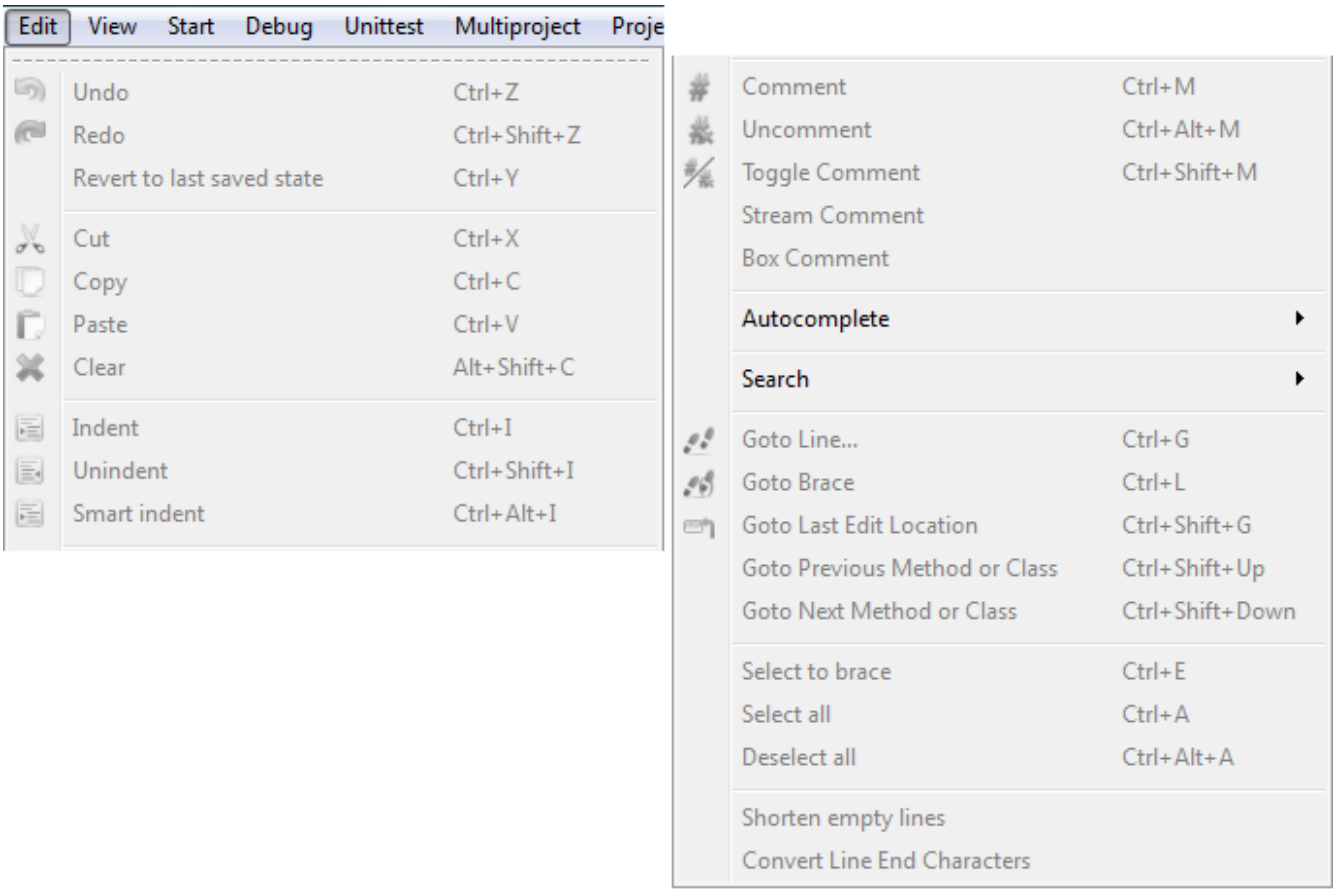
– –

File > **Quit**

Designed to close 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 window sessions will remain unaffected [*see 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   Autocomplete
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
```

- -

Edit > **Undo**

Edit > **Redo**

Edit > **Revert to Last Saved State**

Usual commands to let you change your mid 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 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.

– –

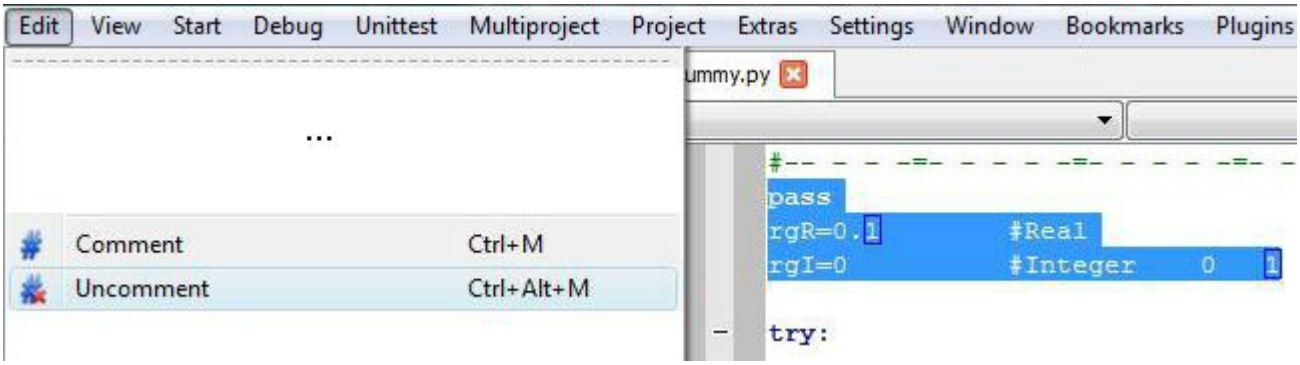
Edit > **Comment**

Edit > **Uncomment**

Designed to add / cancel a leading "#"-Python comment character to the pending line or to a selected block of lines, so to quickly exclude / include their coding effect at one swoop. This "#" addition will take place also with lines possibly already "commented".

Viewpoint

<~> As a mildly odd condition, you have that selecting such a totally un-commented block of lines:



the `Edit > Uncomment` command still keeps remaining active, even though, obviously, ineffective. That's probably due to the presence of some "#" character inside the selected source text.

– –

Edit > Toggle Comment

Aimed at 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.

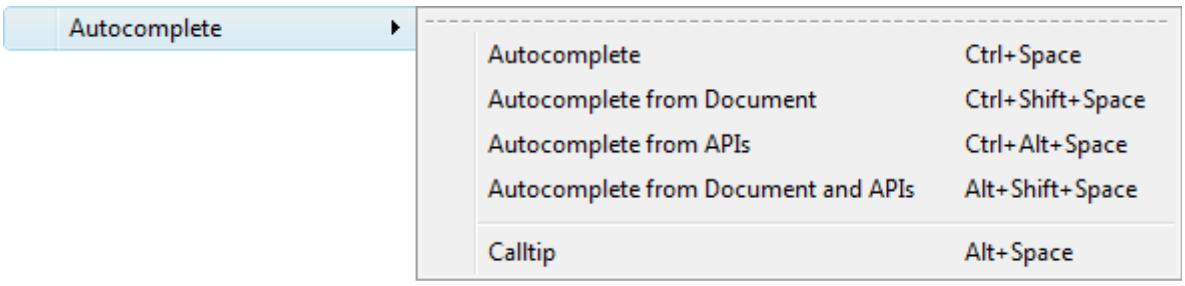
Remark

Eric is potentially a multi-platform, multi-language IDE which, besides Python, could somehow handle other languages such as Ruby [*see section:* Python 2 – 3 (and Ruby), Easy Selection, *in:* {0Lead}].

– –

Edit > Autocomplete

Designed to show a box of sub-menu commands, where to call one of the auto-completion features available for supporting the current source text editing activity.



Here the usage of keyboard shortcuts (e.g.: `Ctrl+Space` for standard Python `Autocomplete`) is particularly convenient. A <`CR`> or <`Tab`> is then required to possibly confirm a text auto-completion action.

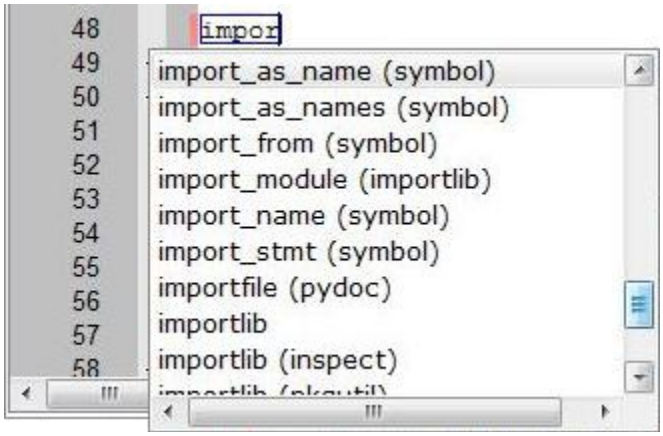
Remark

The initial default auto-completion feature here available is derived from QScintilla[20], as configured with: `Settings > Preferences... – Editor > Autocompletion > QScintilla` [*see*], that is just nothing more than an auto-completion from the very current text document.

Further features could be obtained making use and configuring other APIs with Eric, as with: `Settings > Preferences... – Editor > APIs, Configure API files` [*see*]. This way:

- Select an item from the drop-down "`Language`" list, for instance: `Python3`
- Select an item from the list of installed API files, via "`Add from installed APIs`" button; e.g.: `Python-3.3.api`. Later on, to possibly inspect which are the currently installed APIs, you'll have to select again the related "`Language`".
- Tick the "`Compile APIs automatically`" check-box.

As a consequence, when entering a new Python statemented you'll get this kind of auto-completion aids:



– –

20 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

Viewpoint

<~> We have described this subject at best of our current knowledge and actual experience of use. Very limited indeed, so that we'd welcome some specific remarks from experienced users, so to go beyond this generic description, of limited practical usefulness.

– –

Sub-menu Specifications:

`Autocomplete` Auto-completion feature, as configured on:
`Settings > Preferences... – Editor > Autocompletion`

`Autocomplete from Document`
Text patterns searched on the current source text document

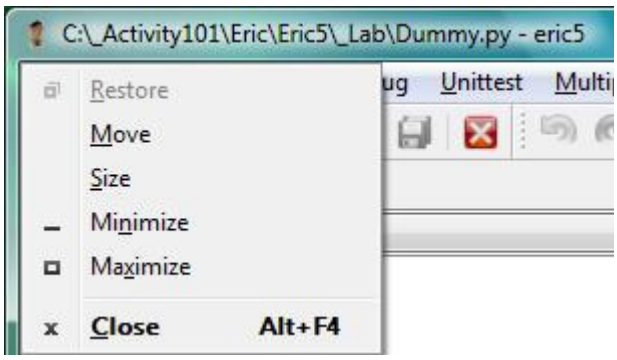
`Autocomplete from APIs`
Auto-completion feature, derived from what configured on:
`Settings > Preferences... – Editor > APIs`

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

– –

And then:

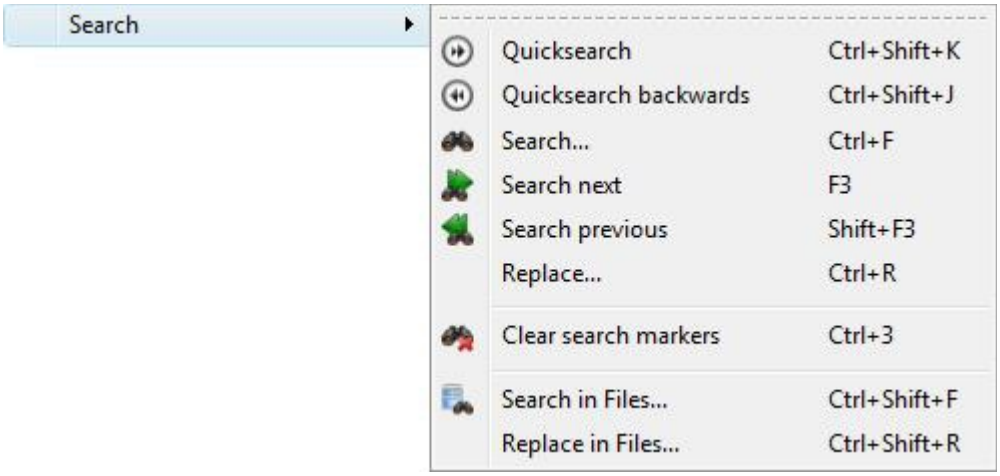
`Calltip` <~> Presumably configurable via: `Settings > Preferences... – Editor > Calltips`, but then no real idea what it's about, therefore ignored. The only visible result obtained was an inconsistent opening of the title-bar menu when entering this command via its "`Alt+Space`" keyboard shortcut.



– –

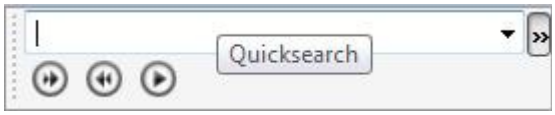
Edit > **Search**

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



Sub-menu Specifications:

`Quicksearch` To search the currently active text edit form for the first occurrence of the pattern entered into the "`Quicksearch`" field, on the tool-bar [*see*], case insensitive, starting from the current text insertion point, forward.



With the auxiliary control buttons: `Forwards`, `Backwards`, `Quicksearch extend`. An `<CR>` on the `Qicksearch` field will then bring the insertion point at the end of the string found, selected.

Remark

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

`Quicksearch backwards`

Same as `Quicksearch` [*see*], but backwards.

– –

`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`

`Search next / previous`
To repeat the same `Search` on and on, same search mode [*cf. former point*].
Conveniently callable also with the usual `F3` short-cut.

`Replace` To add a "`Replace`" text pattern action to the `Find` field.

Action occurs only after explicit pressing of button: "`replace`", or "`and search for next`", or "`replace all`".

`Clear search markers`
To clear 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 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`]. This form is 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.

– –

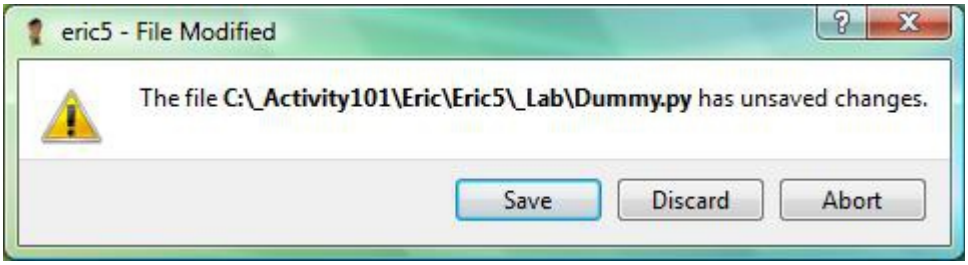
Remark

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

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

Viewpoint

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

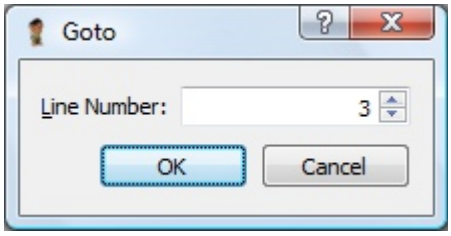


That's because, as with `Debug` menu commands [*see*], also with search actions the synchronization with source text must be regained so to work properly.

– –

Edit > **Goto Line...**

Designed to jump to a given *n*-th line on the currently active Text 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. No action within a comment line, or if no bracket is pointed.

Remark

Some useful hints:

- Matching braces can be highlighted according to command `Settings > Preferences – Editor > Style, Braces` [*see*].
- In such not so infrequent case of a "`](`" contiguous braces, as with: `_vobjmap[_stype](_ent, _data, _bitpos)`, to reach the marching 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[21].

– –

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

– –

21 Just be careful not to misinterpret it as "`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 in the blank—i.e.: empty—lines on the currently active Text Form.

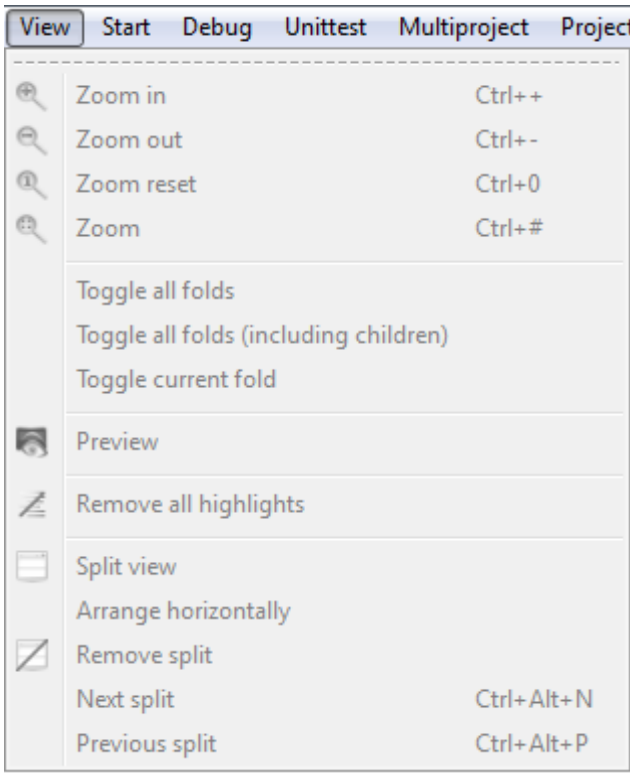
– –

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  Split View         Arrange Horizontally
Remove Split       Next Split             Previous Split
```

– –

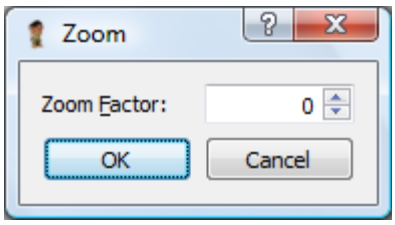
View > Zoom In

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:* {3South}].



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

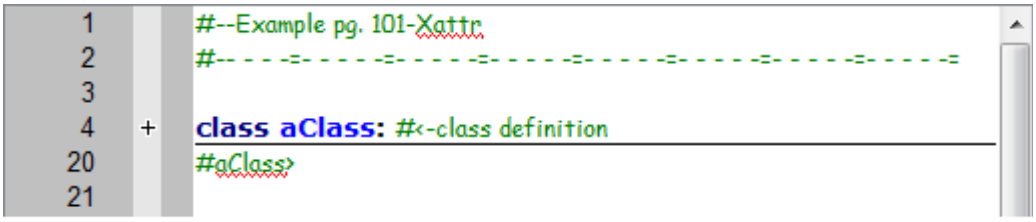
– –

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[22], 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.



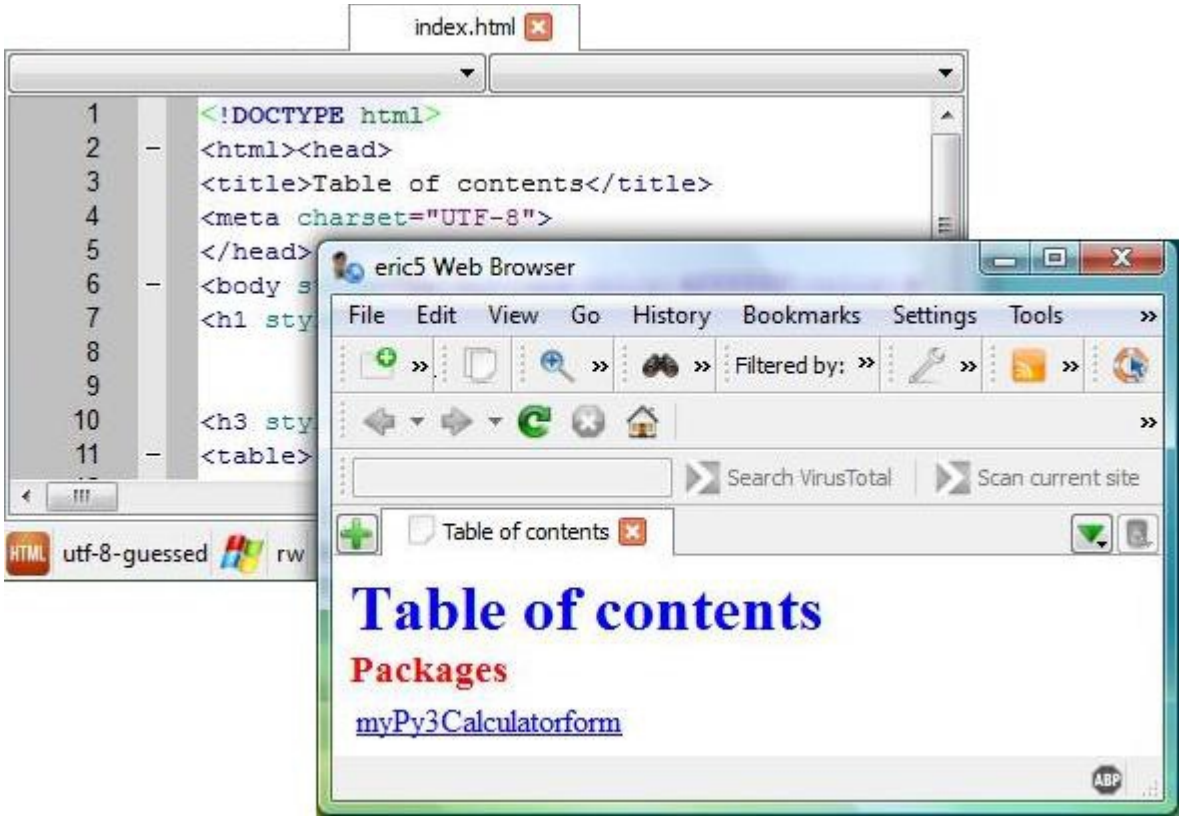
22 A definition derived by the QScintilla "lexer", that is: lexical analyzer.

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[23]. A click on these "+" / "–" symbols has the same toggling effect of these menu commands.

– –

View > **Preview**

Designed for previewing in the Eric web browser [*cf. command:* `Help > Helpviewer...`] a HTML file currently opened into the text edit form, such as, for instance, an automatic "EricDoc" documentation file [*see command:* `Project > Source Documentation > Generate Documentation (eric5_doc)`].



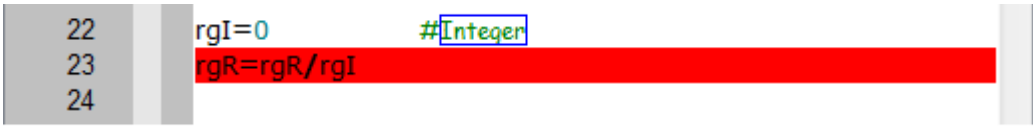
23 Actually these "+" / "–" symbols are configuration dependent, as "`Folding style`" on the "`Margins`" area of the "`Configure editor styles`" [*see:* `Settings > Preferences... – Editor > Style`]. What's here shown is the default "`Plain`" style [*default, indeed: as usual in this* Report].

In case of the usual editing of source text files this command remains disabled[24].

– –

View > **Remove all Highlights**

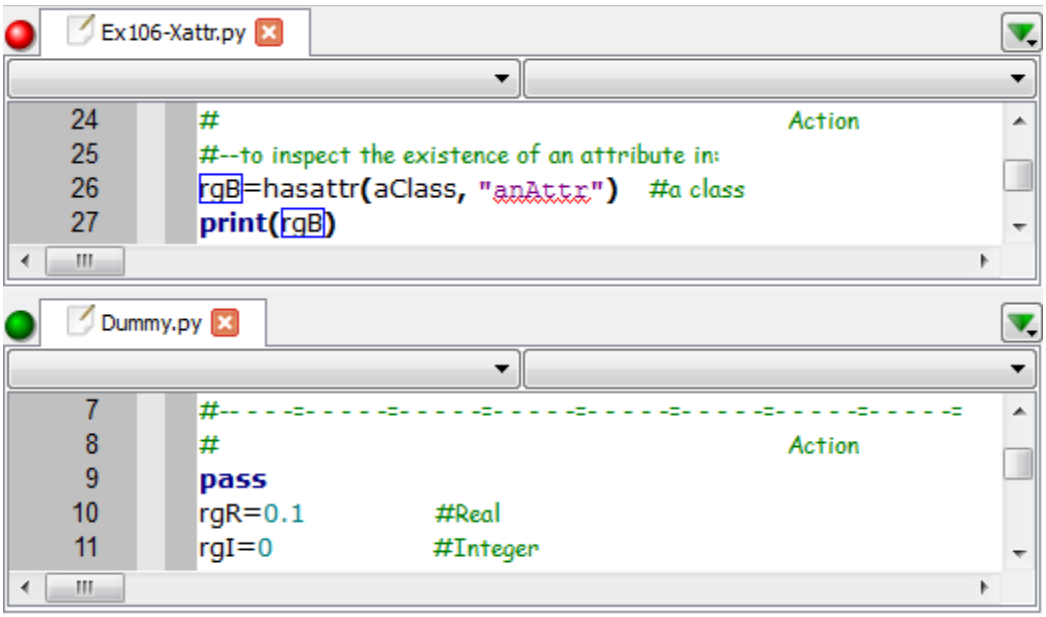
Designed to eliminate all special font-effects, such as the red-highlight for exception rising lines, on the source Text Form.



– –

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`].



24 [*Eric Feature Under Revision*] With next Eric ver. 5.4 different editing files will be automatically associated to different previewers.

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

– –

View > Arrange Horizontally

Designed to set the possible split view mode horizontal, otherwise vertical [*see command:* `Split View`]. 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 command:* `View > Split View`], first closing all 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

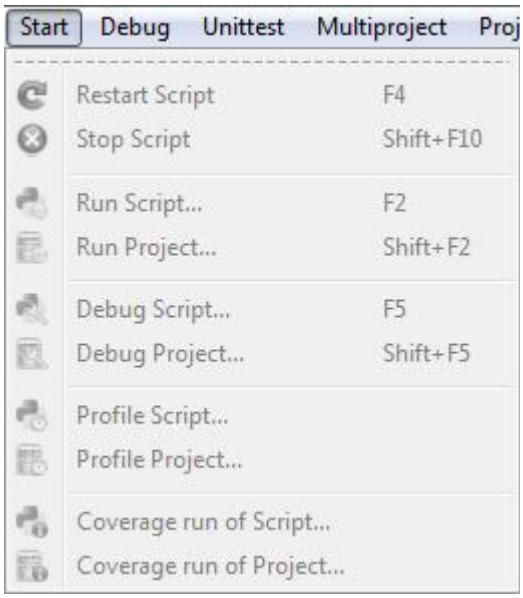
View > Previous Split

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

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

–<>–

Start Command Menu



Command List

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

– –

Start > **Restart Script**

Start > **Stop Script**

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[25], and in Run or Debug execution mode [*see commands:* `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

25 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.

as: `File > Save` [*see*].

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`" on [*see*].

Viewpoint

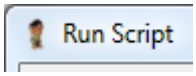
<~> The term *`Script`* is here clearly assumed as referred both to what in all subsequent `Start` menu commands is distinctively called *`Script`* and *`Project`*. We deem it would have been even clearer to say "`Restart Execution / Stop Execution`" or, simply "`Restart / Stop`"[26].

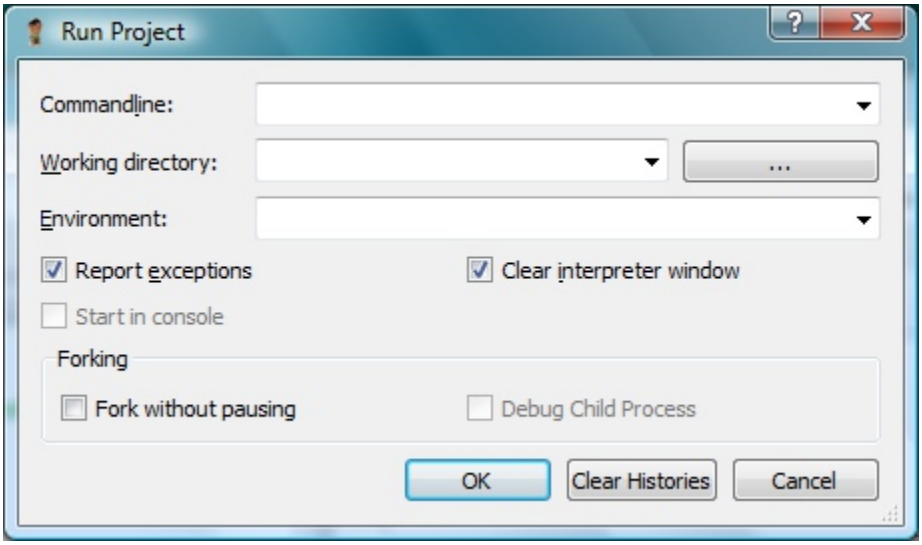
– –

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.





26 We've been informed that these labels will be actually changed with next Eric rev. 5.4.

Specifications:

`Commandline:` List of arguments to be possibly entered into the run command-line, and retrievable via the standard Python "`sys.argv`" call. First argument is automatically assigned to the executing *Main Script*[27] file path [*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 case, differently with the "`Start > Debug`" case [*see*], the handled exceptions are never signaled; whereas un-handled exceptions are signaled anyway.

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

`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, function:* `os.fork()`], the user is not asked which path of execution to follow [*see next:* `Debug Child Process`].

`Debug Child Process`
Upon a process fork, if 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. 跟踪子进程

`Clear Histories`
Button to clear all history values saved in this form.

– –

27 About the *Main Script* concept and role, see above section: Project Command Menu

Remark

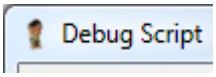
It may be useful here to recall that a script, possibly belonging to a Project, can also be run and tested as a self-standing script, with such executing condition that can be detected via the standard Python test: `if (__name__ == '__main__')`

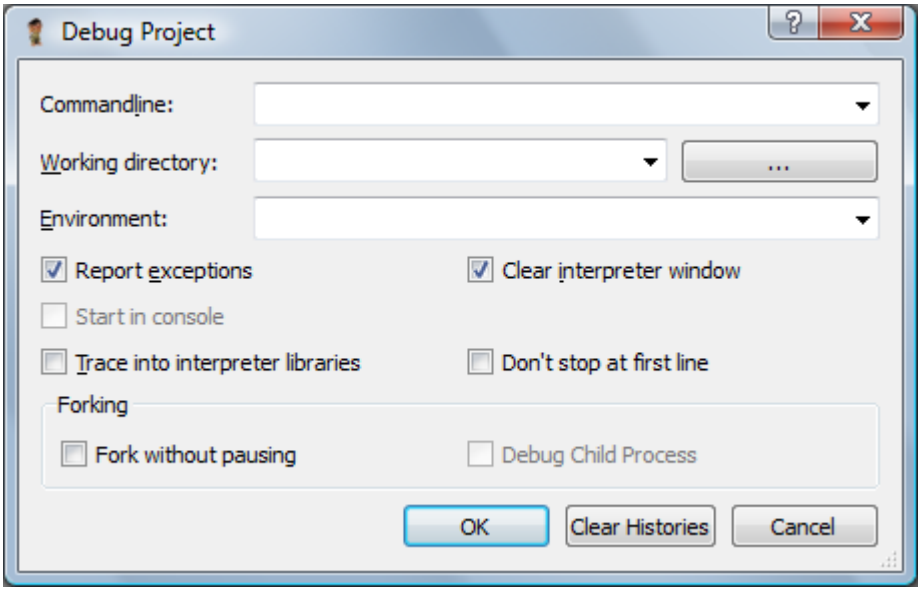
– –

Start > **Debug Script...**

Start > **Debug Project...**

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 [*see also command:* `Debug > Exceptions`] related to possibly both un-handled and handled exceptions — that is: exceptions occurring within a "`try: ...`" block.

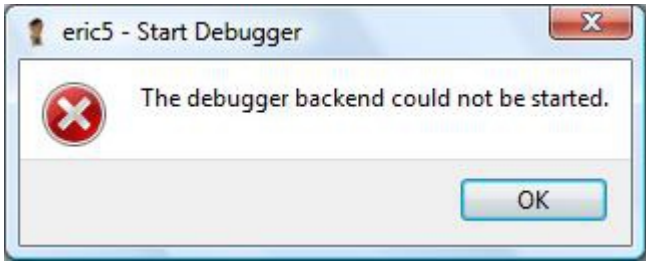
When not checked the exceptions are still effective, but no "`Report`" box will be shown.

`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[28], waiting for user's intervention.

Remark

In case of such an Error Box:



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

– –

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[29], so to collect and record the execution profiling data on disc.

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

29 Here available as a handy interface to the standard Python `profile` / `cProfile` modules.

Profiling data that can then be inspected later on, via:

`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 [*see*];

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

– –

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[30], so to collect and record the code "coverage" data on disc. That is, the data telling which part of the source code has been actually executed and, therefore, tested during a run session. Comments and empty lines are ignored[31].

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

`Erase coverage information`
: Check-box, where you have the choice whether to keep or to 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 means that the so far un-covered (i.e.: not-executed) statements can possibly switch their status to covered.
  - Erasing this information means to get a coverage status exclusively as the result of the last run. An advisable choice in case of source changes[32].

– –

30 <。> `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/`]

31 <~> Whereas, as already spotted, the treatment of "`pass`" statements is rather ambiguous. Anyhow this is Python matter, not Eric's.

32 But deliberately 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 inspected later on, by means of:

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

`(^) Show > Code Coverage...`
: Right-click context menu command, for a particular module on a Project-Viewer [*see*];

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

`Bookmarks > Next / Previous Uncovered Line`
: Source text edit menu commands [*see*].

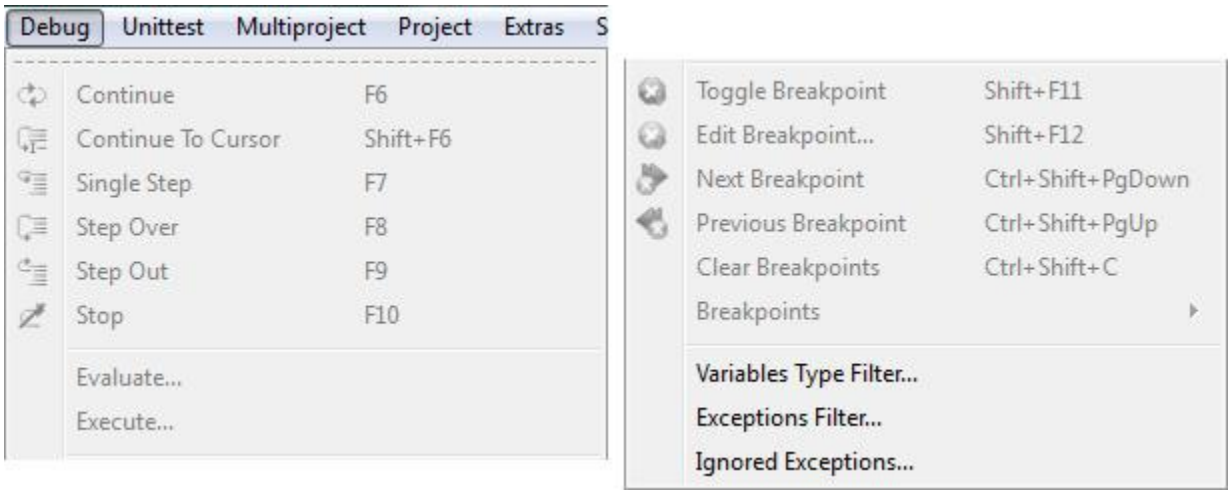
– –

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 [*see:* `Window > Bottom Sidebar`]. Same action can be carried on operating directly into the very Interactive Shell Form[33].

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>`"[34].

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*].

– –

33 Where no operative difference have we found, but the fact that working on the Interactive Form is easier.
34 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 [*see:* `Edit Breakpoint...`].

Remark

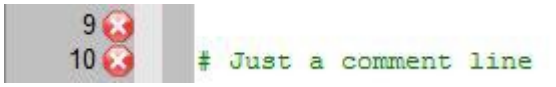
<!> It's worth knowing that the set of breakpoints possibly defined when working on a Project can be saved and then restored making use of the `Project > Session` commands [*see*].

– –

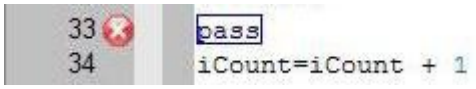
Viewpoint

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:

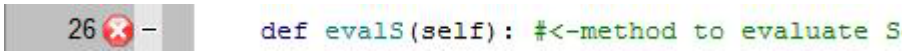
- Empty, or comment lines.



- "`pass`" statements, as they are execution-ignored by Python[35].



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

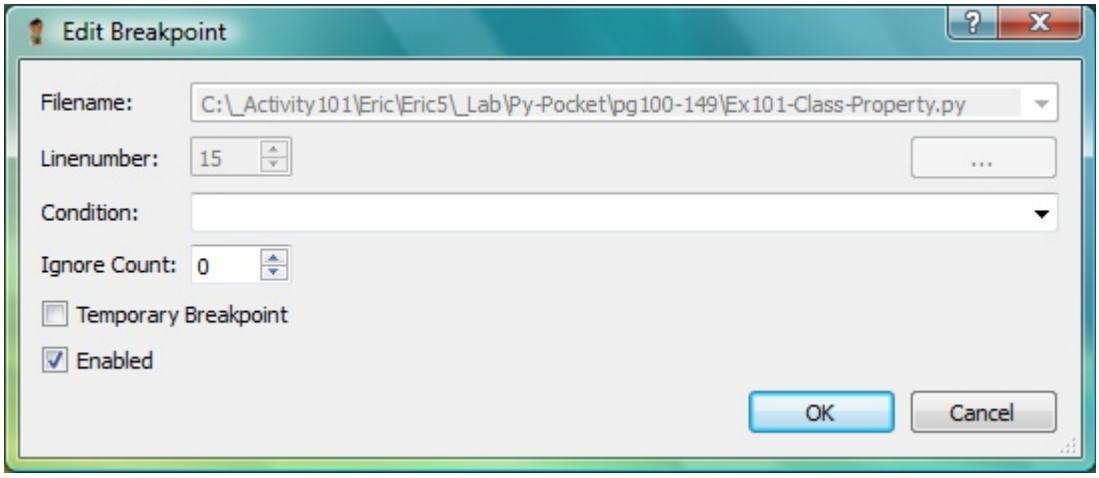


– –

35 <~> We do not agree, as we think "`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.



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 Form.

– –

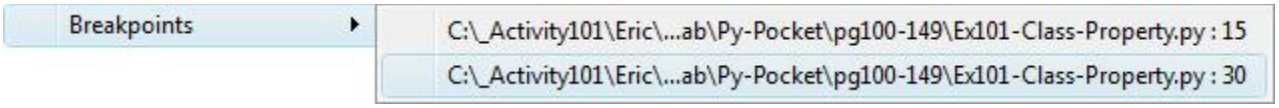
Debug > **Clear Breakpoints**

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,

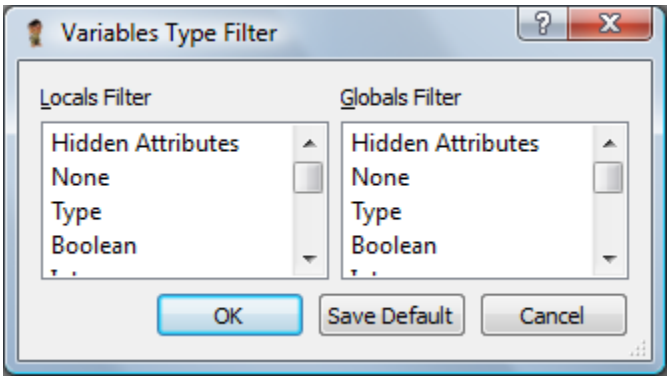


so also to possibly move the cursor onto one of them.

– –

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[36] exceptions (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...`].

– –

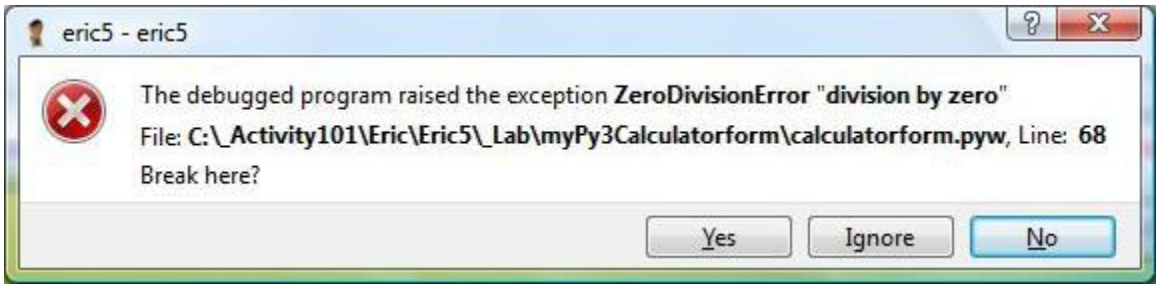
Debug > **Ignored Exceptions...**

Designed to show a dialog box where to enter the code of which one of the Python handled[37] exceptions (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:

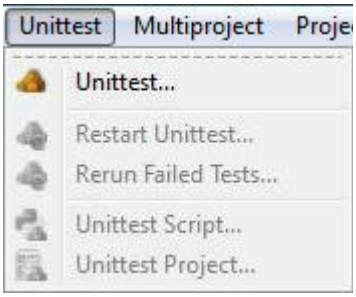


the involved exception code will be automatically added to this same `Ignored Exceptions` list [*see also the "*`Report exceptions`*" check box on the "*`Start > Debug`*" command*].

-<>-

36 That is, exceptions occurring within a "`try – except`" block.
37 That is, exceptions occurring within a "`try – except`" block.

Unittest Command Menu



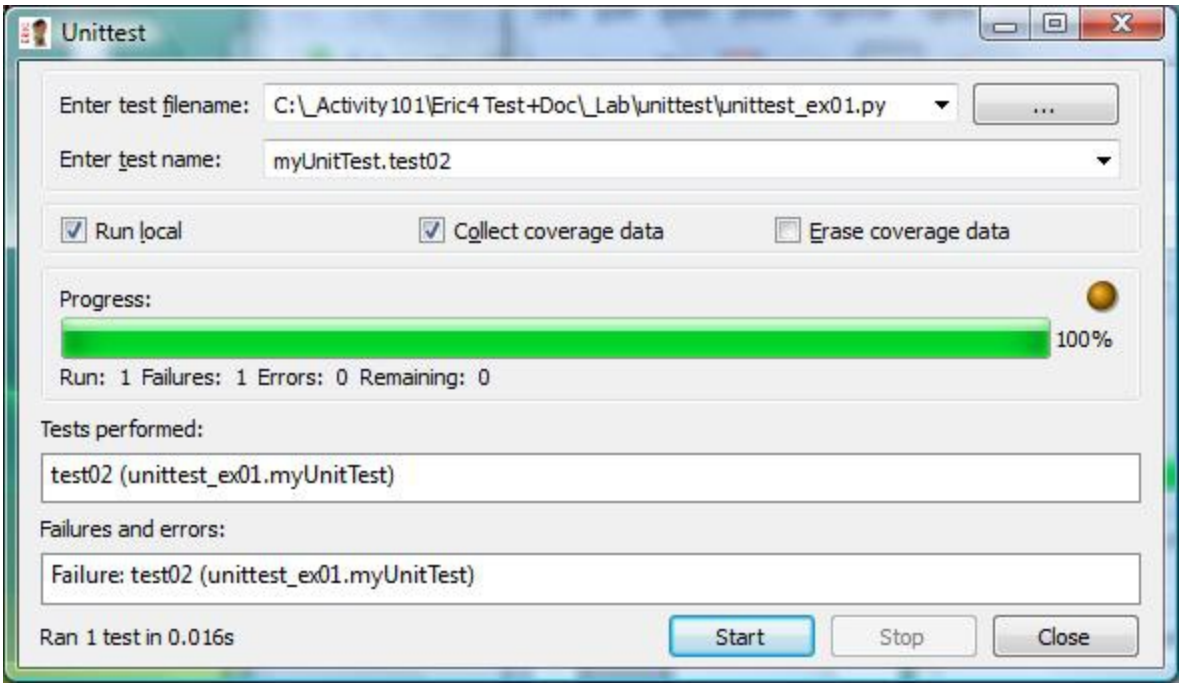
Command List

```
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.

`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[38].

`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 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...**

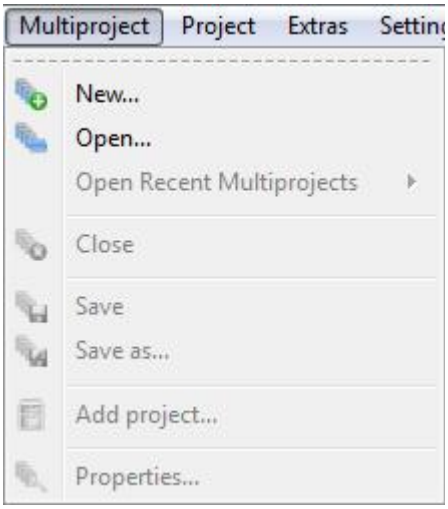
Unittest > **`unittest` Project...**

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

-<>-

38 Functionality that 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 a Multi-Project as a collection of different Projects that, for some reason, it's convenient to regard as a single macro-entity.

In a Multi-Project there is a special "*`Master Project`*", defined as the first to open. It's a concept corresponding to that of "`Main Script`"[39] defined in a Project [*see next section:* Project Command Menu] as the first source module to open and execute.

– –

39 So corresponding that it would be reasonable to adopt a similar denomination, as "Main Script" and "Main Project".

 Indeed, this naming is due to be changed next rev. 5.4.

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.

Remark

These the commands hereafter described in detail, as specific of the Multi-Project structure.

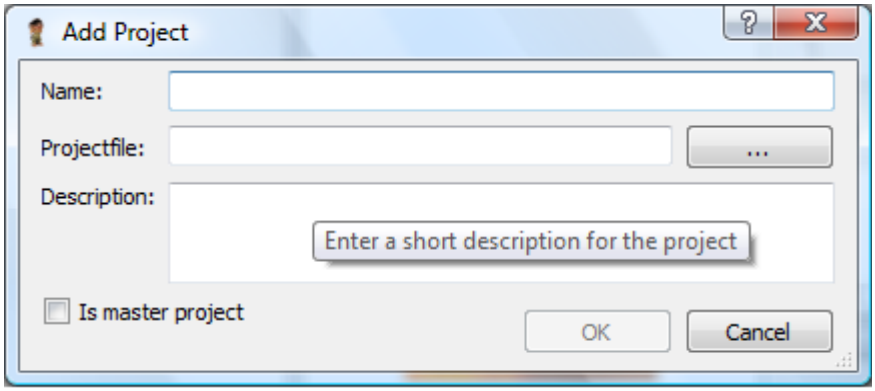
`Add Project...`

`Properties...`

– –

Multiproject > **Add Project...**

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

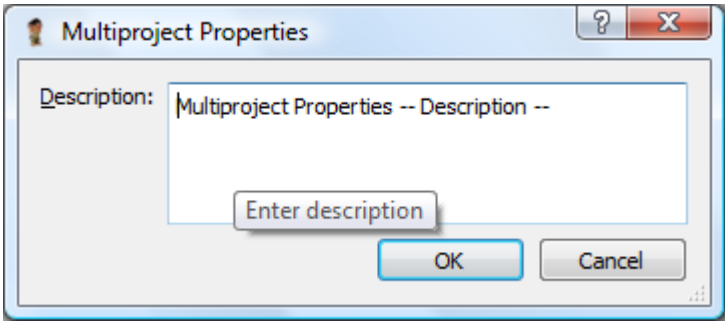


Dialog box assumed self-explanatory enough not to require any further description.

– –

Multiproject > **Properties...**

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

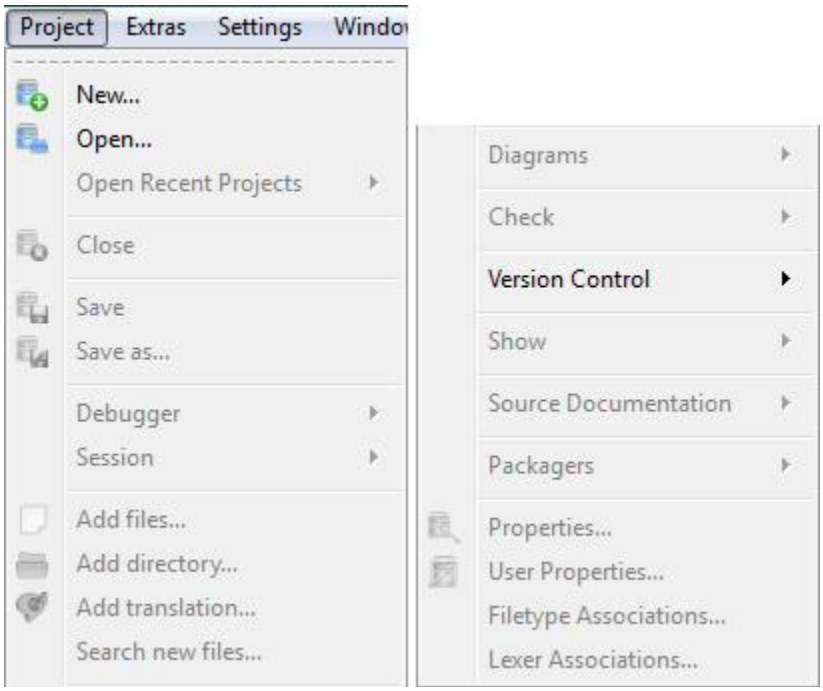


Remark

Such "`Description`" property is here to be considered simply as the initial one of a 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 end, 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. command 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 here:

`Project Directory`
: Project's host directory, hosting, in particular:

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

  `_eric5project` — A Project management sub-directory, automatically created to keep some complementary XML files.

Remark

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

– –

`Main Script`[40]
: Project's entry script, where execution will start. It's a concept corresponding to that of the "`Master Project`"[41] in a Multi-Project [*see*].

`Version Control System`
: Property automatically asked at a new Project inception about the possible adoption of a Version Control System (VCS)[42]—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.

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-synchronization action required for debugging after the possible editing of a source script [*see*].

– –

40 A term not so infrequently used interchangeably with *Module* [*cf.:* Glossary, *in:* {Map}].

41 <~> So corresponding that it would be reasonable to adopt a similar denomination, such as: "*Main Script*" and "*Main Project*", instead of "*Master Project*".

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

Project > **New...**

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.

Besides that, the `Project > New...` command will show a special "`Project Properties`" form, the same one of the `Project > Properties...` command, and there described [*see*].

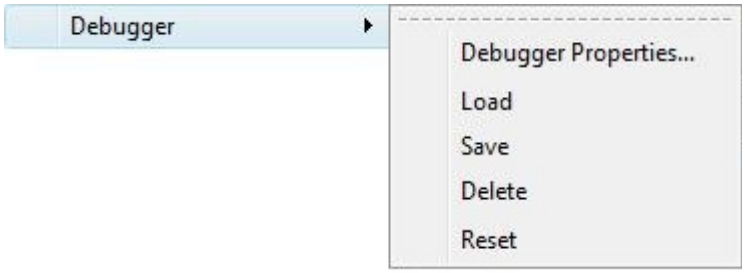
Remark

All the other commands, as specific of the Eric Project structure, will be hereafter described 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[43].



43 [*Eric Feature Under Revision*] News have we recently got that this command will be modified with Eric ver. 5.4 so to appear friendlier, in particular so to have the `Load` and `Delete` sub-commands disabled when no file has been actually `Saved`.

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 "`_eric5project`", on a XML "*`myProject`*`.e4d`" file.

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`" 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:* {0Lead}].

– –

Project > Debugger > **Load**

Project > Debugger > **Save**

Project > Debugger > **Delete**

Project > Debugger > **Reset**

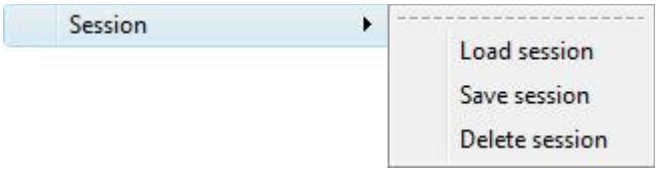
Suit of commands designed to manage the specific set of Debugger Properties possibly defined for the current Project.

– –

Project > **Session**

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

Operatively a Project Session is such a XML text file: `<myProject>.e4s`, automatically named after the current Project and located into its standard project management sub-directory `_eric5project`.



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

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".

Viewpoint


A rather useful little feature, soon meant to be improved this way[44]:

`Load Session` Would be disabled when there's no saved session to load;

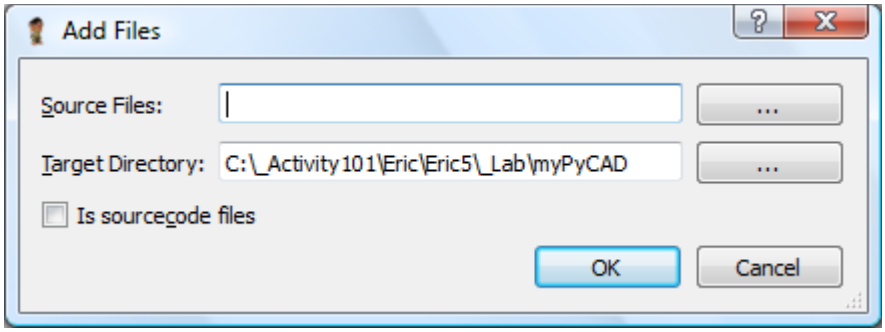
`Delete Session` Would be disabled when there's no saved session to delete.

– –

44 Presumably with Eric ver. 5.4.

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 a possibly different extension

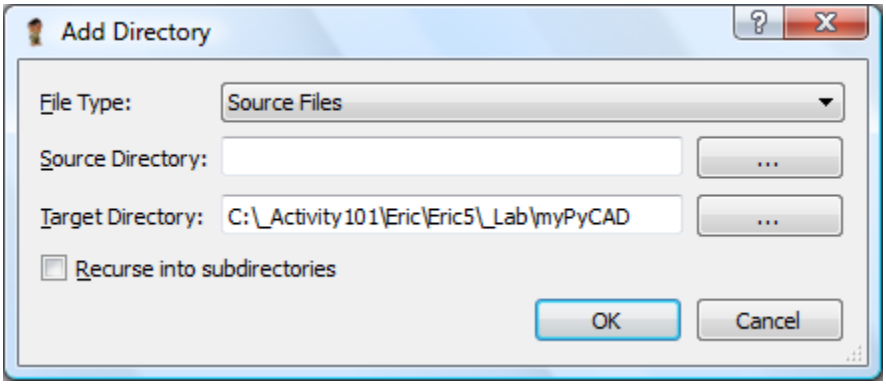
Remark

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".

– –

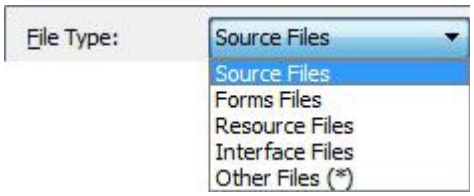
Project > **Add Directory...**

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



Specifications:

`File Type:` File-type filter, for possibly selecting the type of files



to be added to the project

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

`Target Directory:`
Directory to associate to the project, where current new files are to be added; possibly copied if it is different from the source directory

`Recurse into subdirectories`
Addition 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 into a given language. A feature mainly based upon 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 feature are also with commands: `Extras > Tools (Select Tool Group = Builtin Tools)> Qt-Linguist..., > Translations Previewer...` and the context menu of *Project-Viewer:* Translations on the L-Pane [*see in:* {4West}].

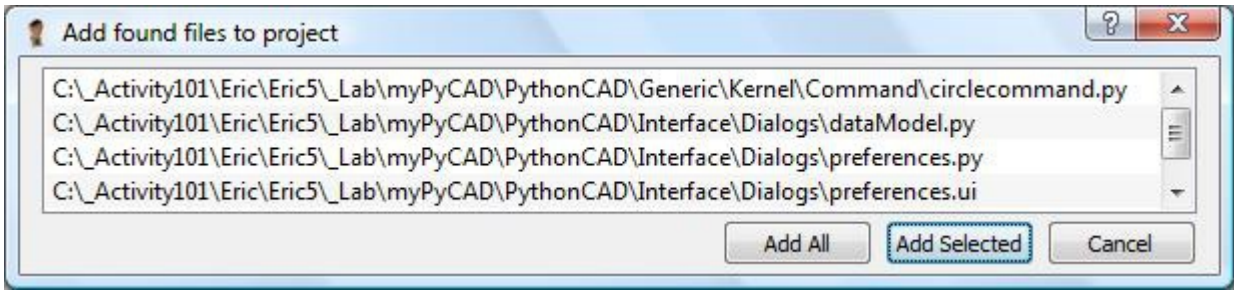
Viewpoint

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

– –

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

Designed to display an "`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. This search is recursively extended to all Project's sub-directories known to Eric, with possible sub-directories currently not assigned to the Project that will not be searched.

Remark

The directory searched is the "`Project Directory`", as set on the "`Project Properties`" form [*see command:* `Project > Properties...`]. Note that in this "`Add … Files`" form you'll see listed all files there found, no distinction whether already belonging to the project or not; and no sub-directory.

– –

Project > **Diagrams**

Designed to call a sub-menu of commands

aimed at creating a symbolic diagram, that is a graph-based representation of the Python Application under development in the current Project.

Command List

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

– –

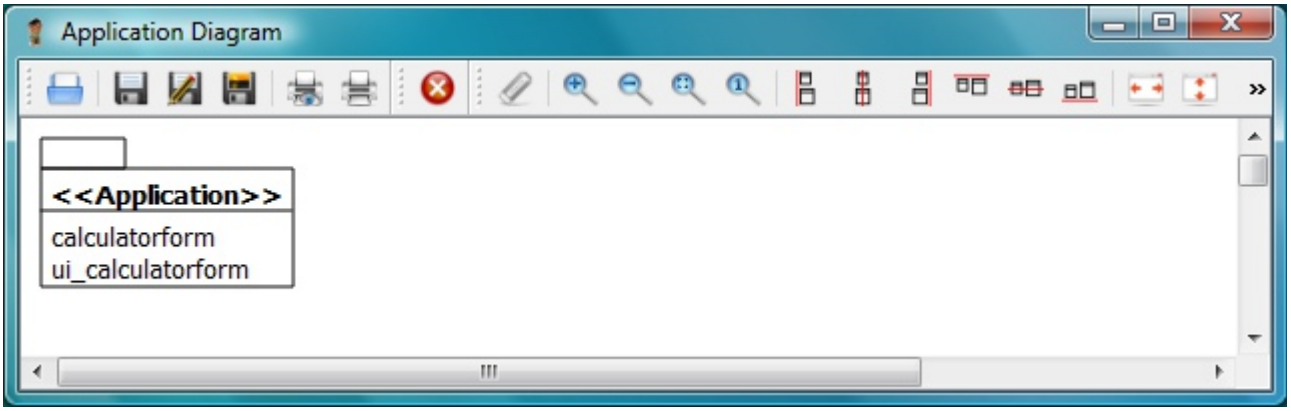
Viewpoint

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

– –

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

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



A diagram that can be saved as an "Eric5 Graphics file"[45], 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 "Eric5 Graphic file", of type "`*.e5g`", so to be then possibly further edited[46] [*cf. former:* `Diagrams > Application Diagram...`].

– –

Project > **Check**

Designed to call a sub-menu of commands

aimed at performing some standard source checks on the current Python Project.

Viewpoint

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

Command List

`PEP 8 Compliance...` `Syntax...` `Indentations...`

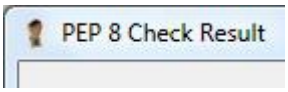
– –

45 <~> Here too, as default target location, we'd suggest the very Project's directory, instead of the current Python's installation directory.

46 <?> At our first opening of a just saved Eric5 Graphic file, we've seen this rather baffling title bar: "`Illegal Diagram Type`".
<~> Here too, as with former `Application Diagram... command` [*see*], we'd suggest a better choice for the default directory.

Project > Check > **PEP 8 Compliance...**

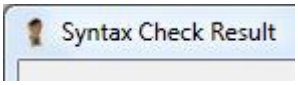
To quickly scan current Python Project for compliance with the so called PEP 8 Python Project's code style guideline[47]. The dedicated "`PEP 8 Check Result`" window is for the management of this check, and for the display of consequent results [*see*].



– –

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

To quickly scan current Python Project for syntax errors[48], possibly displayed on a "`Syntax Check Result`" window.



– –

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

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

Remark

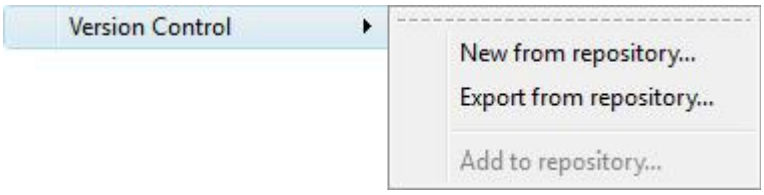
"*`Tabnanny`*" is a standard Python service module, aimed at the "*Detection of ambiguous indentation*". But, having to do with semantics, be warned that this test is not, and couldn't possibly be, a sure bet.

– –

47 Python Enhancement Proposals (PEPs), PEP 8 -- "Style Guide for Python Code", 05-Jul-2001.
48 Making use of "PyFlakes", the popular checker of Python programs.

Project > **Version Control**

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

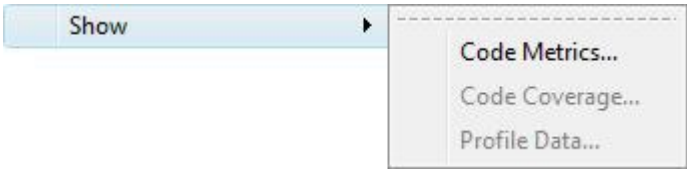
This is a self-standing subject of particular relevance, here just hinted at as assumed off the scope of this Report [*see:* Scope of this Report, *in:* {0Lead}].

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[49] as it is basically valid for this Eric 5 case 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.

Command List

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

– –

49 See URL in: Foreword, *What & Where in Internet*

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

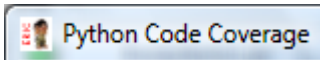
Designed to show a "`Code Metrics`" window,

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*]. Disabled if no "`Coverage`" run executed.

– –

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.

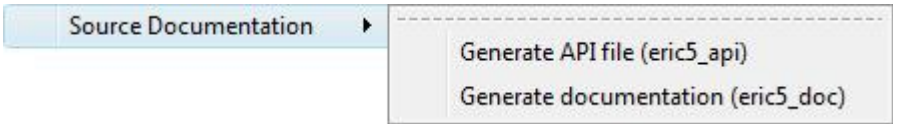
Viewpoint

Impressive rich set of data, that we'd like to interpret better than we currently are able to do.

– –

Project > **Source Documentation**

Designed to call a sub-menu of commands



aimed at generating "`API`" and "`Documentation`" files derived from current Eric source Project [*cf. command:* `Extras > Wizards`].

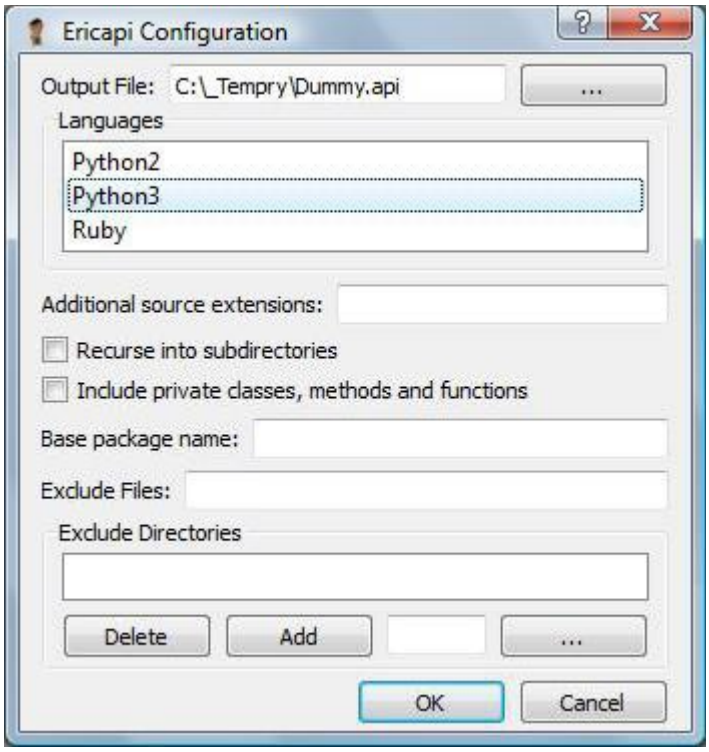
Command List

`Generate API File (eric5_api)` `Generate Documentation (eric5_doc)`

– –

Project > Source Documentation > **Generate API File (eric5_api)**

Designed to show such an "`Ericapi Configuration`" control box,



meant to set up the desired parameters for the "`eric5_api`" script, and then 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.

– –

Remark

This very "`eric5_api`" script is part of a standard Eric installation and can be run also autonomously, independently from Eric [*see: "`*.bat`" files, in: `C:\Python3x` directory, after a successful Eric installation*], and will generate "`Output File`" of "`*.api`" type[50], which are text files with a contents of this kind:

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

Viewpoint

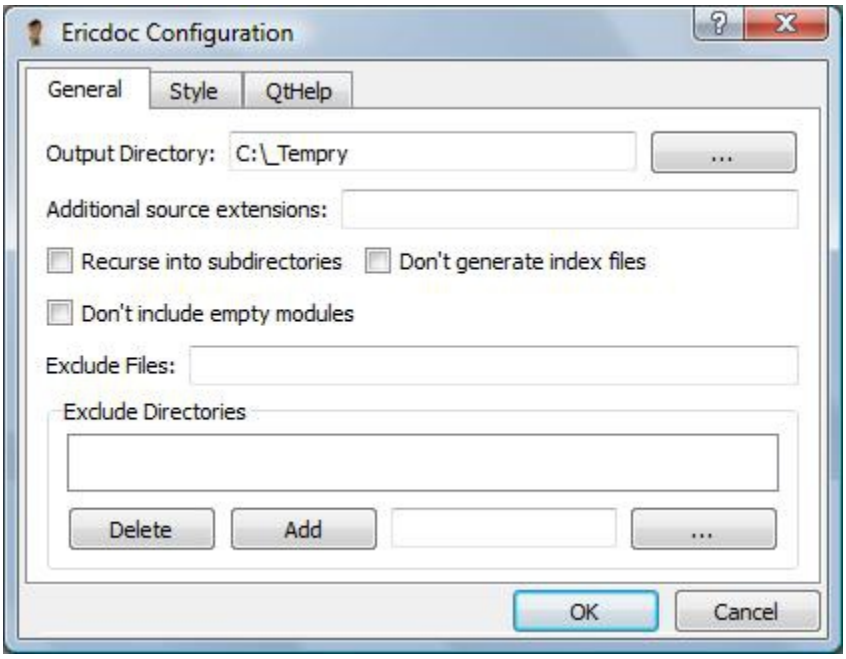
A subject connected with commands `Edit > Autocomplete` and `Edit > Autocomplete > Calltip` [*see*], here just hinted at as assumed off scope for this Report [*see:* Scope of this Report*, in:* {0Lead}].

– –

50 <~> We'd suggest a better choice for the default directory of this API – "Application Programming Interface" file. That is the Project's, instead of the Python's installation directory.

Project > Source Documentation > **Generate Documentation (eric5_doc)**

Designed to show this "`Ericdoc Configuration`" control box:



aimed at generating automatically a set of documentation files derived from the source items contained into the current Eric Project's host directory[51].

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

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 box are assumed clear enough not to require any further description. Just a suggestion: set the check-box "`Recurse into subdirectories`" if you want to scan all your Project's directory tree.

51 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.

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 > Tools > Select Tool Group`].

This sub-menu results enabled only for "`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 the scope of the current Report [*see:* Scope of this Report*, in:* {0Lead}].

Command List

`Create Package List` `Create Plugin Archive` `Create Plugin Archive (Snapshot)`
`Use cxfreeze`[52] [*optional, see:* `Plugins > Install Plugins...`]

– –

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

Project > Packagers > **Create Package List**

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

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 "snapshot").

– –

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

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

Viewpoint

<!> And this last is really a remarkable point, worth of note.

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

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

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:* {0Lead}], but also so relevant that we've carried on a dedicated investigation, and produced two Reports[53] on the subject:

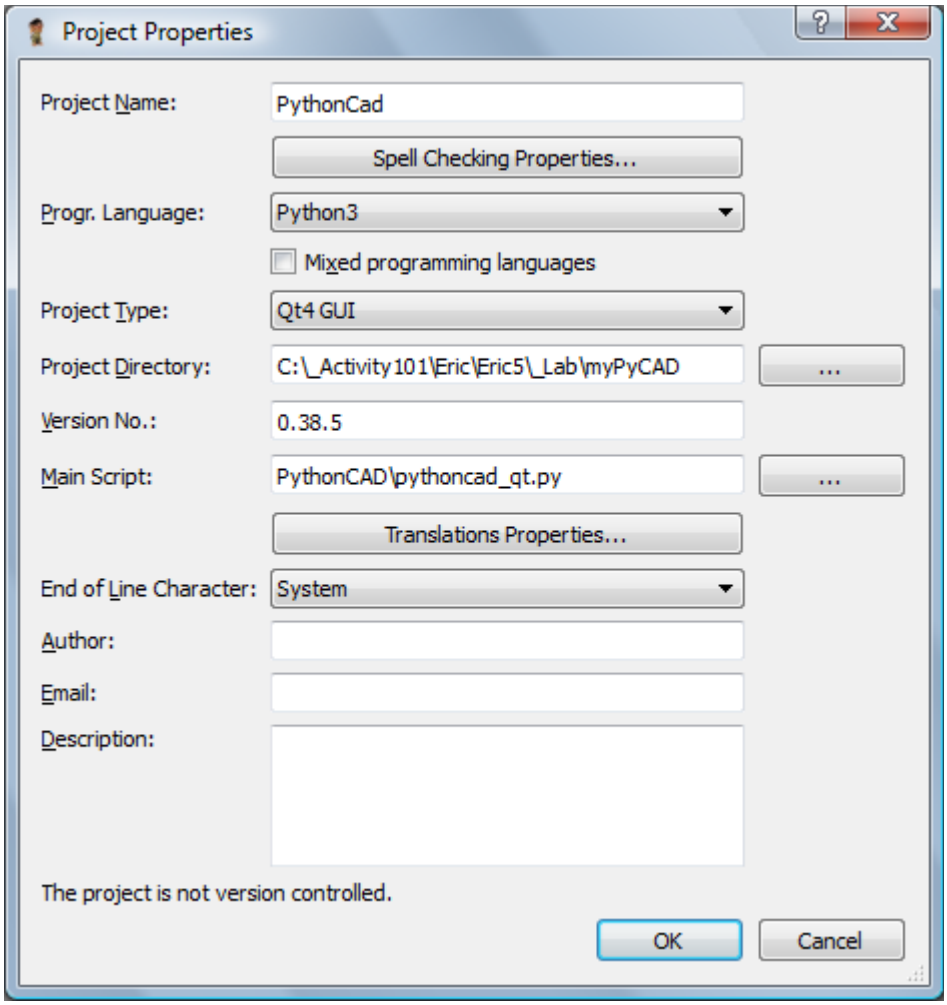
- One about the limited "`cxfreeze`" technique, as currently available with Eric, and as here hinted at;
- And one about the more powerful cx_Freeze "`distutils`" technique, here envisioned as a considerable enhancement for Eric.

– –

53 Both unpublished, so far. If interested, 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 appearing at a new Project creation [*cf.:* `Project > New`].



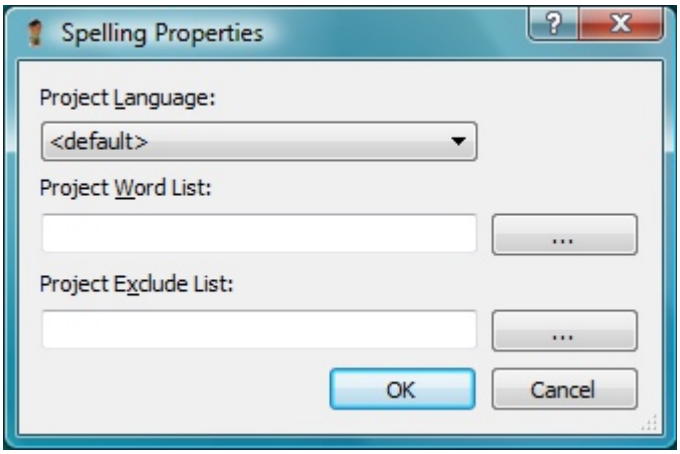
Specifications:

`Project Name` Here the original Project name, as assigned at `Project > New...` creation [*see*], can be 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 possibly eliminated by hand.

`Spell Checking Properties...`

A button to show such "`Spelling Properties`" box,



where to possibly modify locally some of the base properties as set via `Settings > Preferences... - Editor > Spell checking` command [*see*]. Properties setting that are related to the spell checking feature, as with "`Extras >`" main menu commands and with text edit context "`Spell (^)`" menu commands [*see*].

Viewpoint

<. . .> Here we'd welcome some specific remarks from experienced users, so to go beyond this generic description, of limited practical use.

– –

`Progr. Language`

A drop-down list offering a choice amongst all possibly available languages, such as Python 2, Python 3 and Ruby. That is, beyond Python 3, which is mandatory [*see:* Python 2 – 3 Compatibility*, in:* {0Lead}*, and also* `Shell (^) Start` *context command, in:* {3South}].

`Mixed programming languages`

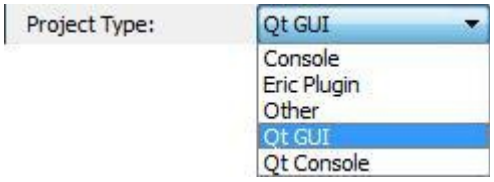
A check-box for accepting as valid Project modules all file types associated with any of the known languages, otherwise the only one associated with the selected `Progr. Language` parameter [*see*] will be accepted.

Remark

It's here worth recalling that the Eric IDE is primarily aimed at the Python language, but can be also used with the language Ruby. Feature, anyhow, here just hinted at, and not 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 execution will start

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

Viewpoint

<。。。> Here we'd welcome some specific remarks from experienced users, so to go beyond this 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

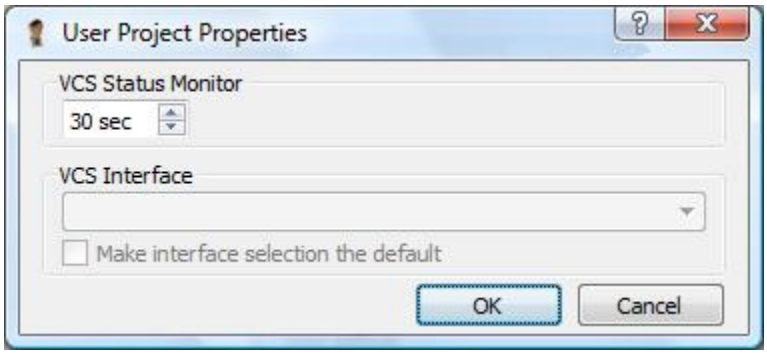
Remark

<!> The bottom-caption: "`The project is not version controlled`" 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...**

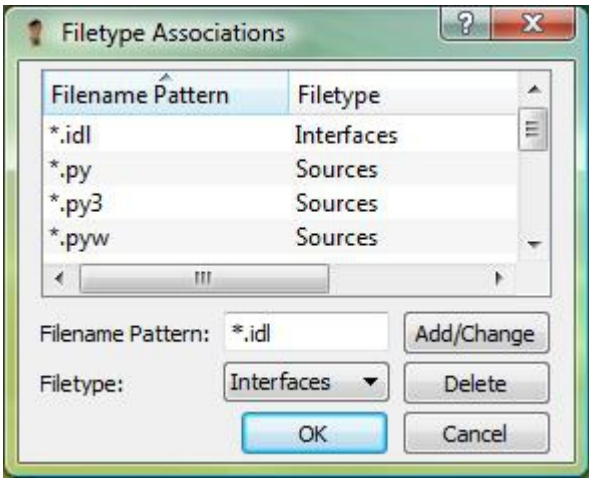
Designed to show a "`User Project Properties`" box just aimed at displaying some info about the Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted for the current Project [*see command:* `Project > Version Control`].



– –

Project > **Filetype Associations...**

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

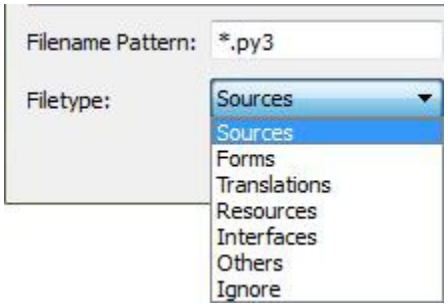


More precisely this command can associate file-name "Patterns"[54], a concept including such sheer "`*.py`" file extension as a particularly common and simple case.

54 Also known as: "wildcard pattern".

Viewpoint

Notable the "`Filetype`" drop-down list [*see*], showing a list of the most common file-types in use.

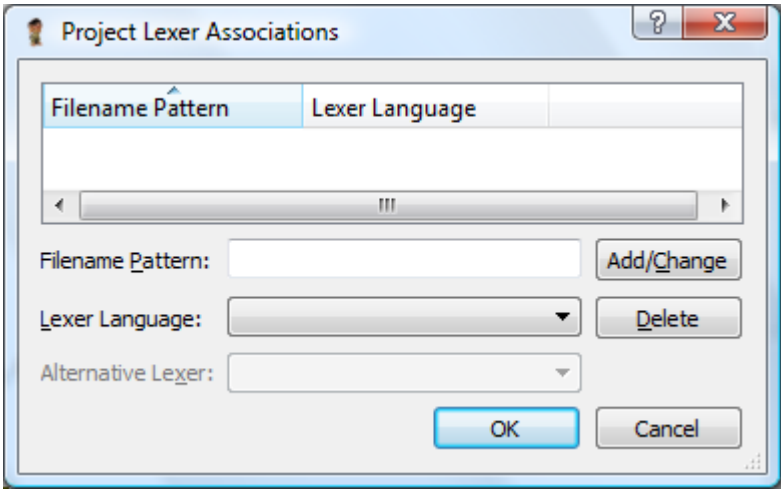


<.?.> Anyhow to complete the description of this command we should also know which are the Eric features affected by such association. Further investigation is thus here required.

– –

Project > **Lexer Associations...**

Designed to show such a "`Project Lexer Associations`" control form,



<.?.> about which all we know is that this "Lexer" thing has to do with the QScintilla lexical analyzer, and is meant to override some global setting—unknown to us—on a per Project basis. Not enough even to try an exploratory test.

– –

Viewpoint

<。?。> [*Hic sunt leones*[55]] We'd like to take such a case as an opportunity to remember Mr. Reader that one of the purposes, if not the primary one, of all these Eric Reports of ours is to encourage a serious technical dialog among all parties involved, to the common benefit. Here, for instance, are some suggested *Themes* to stimulate a discussion.

T1) When encountered an unknown / obscure command / option / feature, the wisest thing to do first is to skip and ignore it. Maybe then, in future, climbing the "learning curve", or talking with a colleague …

T2) Here everybody knows, or should know, what an Eric "Lexer Associations" is, and users who don't should go elsewhere to find an answer. Or change trade / product.

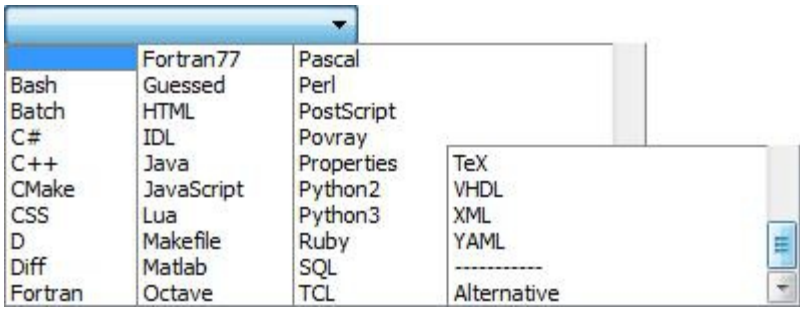
T3) When encountered an unknown / obscure command / option / feature, there should be a practical way for knowing at least what it is all about, so to be capable of evaluating whether it is the case to make some learning efforts, or not. And the best "practical way" is: *support*, not necessarily free.

T4) – *Nobody's Perfect!* [Billy Wilder's "Some Like It Hot"].

– –

Viewpoint

Plus then, the "`Lexer Language`" control drops down this rather impressive list of languages:

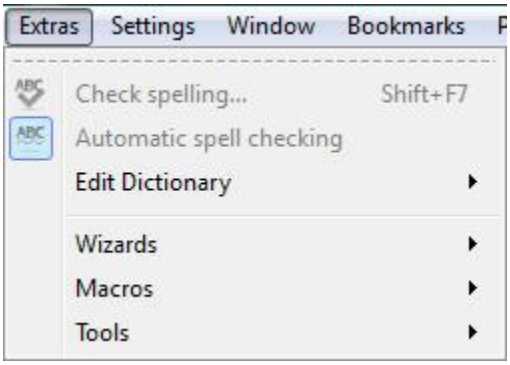


<~> about whose role and meaning we'd like to know more.

-<>-

55 Latin for "*Here there are lions*", reminiscent of when, in ancient maps from the Roman era, African regions about which there was no information were so identified. The reason for the inscription was to warn the potential traveler of dangers that he may encounter in the unmapped areas.

Extras Command Menu



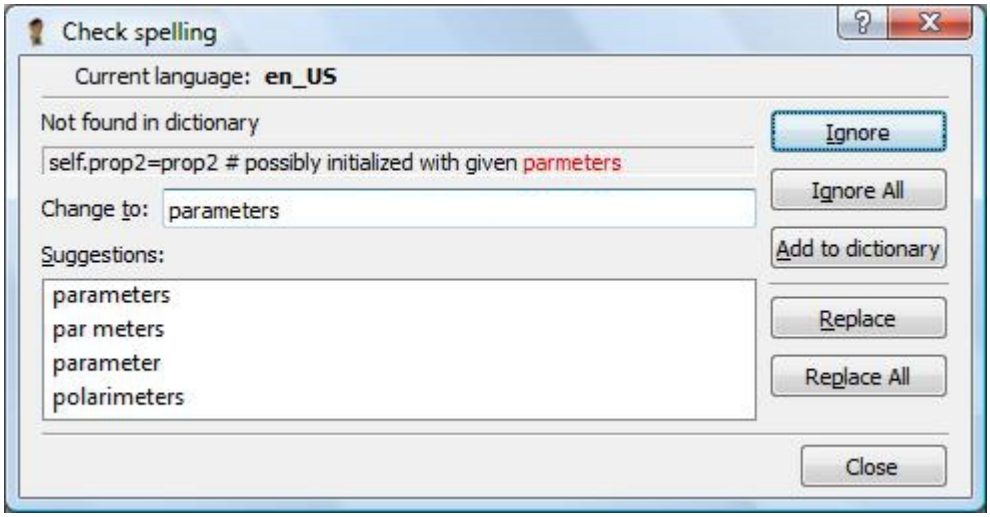
Command List

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

– –

Extras > **Check Spelling...**

Designed to show this "`Check spelling`" box:



where to check and manage the spelling issues of text written on the current Text Form, typically the

comments. This command requires the presence of the PyEnchant spell-checker toolkit [*see also:* `Settings > Show External Tools`]. With such toolkit not installed, this command results "dim"-disabled.

`Add to dictionary`

Button provided to possibly add to the Dictionary in use a word here declared as "`Not found in dictionary`". There are two of such Dictionaries, a global and a Project's [*see next* ***Remark***], both plain text files and, as such, easily examinable and, possibly, editable by means of an ordinary text editor.

Remark

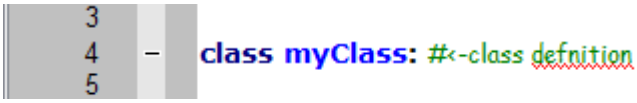
Spell-check options—notably: global Language and Dictionary—can be managed with menu command `Settings > Preferences... - Editor > Spell checking` [*see*]. Where, in case of PyEnchant not installed, you will be informed that "`Spell checking with PyEnchant is not available`". Default storage location: "`<User>\_eric5`" directory.

Language and Dictionary can be also defined at the Project level, with menu command `Project > Properties... - Spell Checking Properties...` [*see*]. Default storage location: Project's host directory.

– –

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*], otherwise unchanged.



Remark

For actual spell editing see menu command: `Extras > Check Spelling...`

– –

Extras > **Edit Dictionary**

Designed to call a sub-menu of commands aimed at managing the Project's and User's dictionaries.

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**

Designed for editing the named dictionaries, no action in case of a dictionary not defined[56].

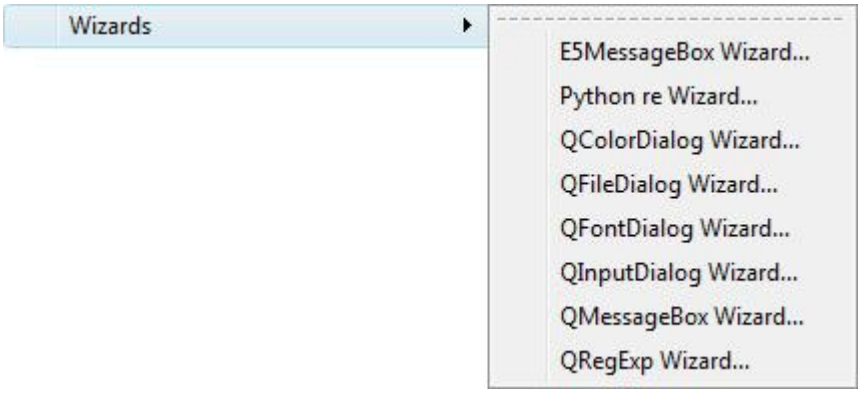
About the creation, location and use of such dictionaries, refer to the above command: `Extras > Check Spelling...` [*see*].

– –

56 Feature under revision: commands will simply become disabled for dictionary not defined.

Extras > **Wizards**

Designed to call this sub-menu of "wizard" commands,



aimed at generating "`API`" and "`Documentation`" files derived from Eric source Projects [*see command:* `Project > Source Documentation`].

Remark

The commands shown in this `Extras > Wizards` sub-menu correspond to last eight "`Wizard`" items in the `Plugins > Plugin Infos...` list [*see*].

– –

Command List

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

Viewpoint

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

– –

Extras > Wizards > **E5MessageBox**

Extras > Wizards > **Python re**

Extras > Wizards > **QColorDialog**

Extras > Wizards > **QFileDialog**

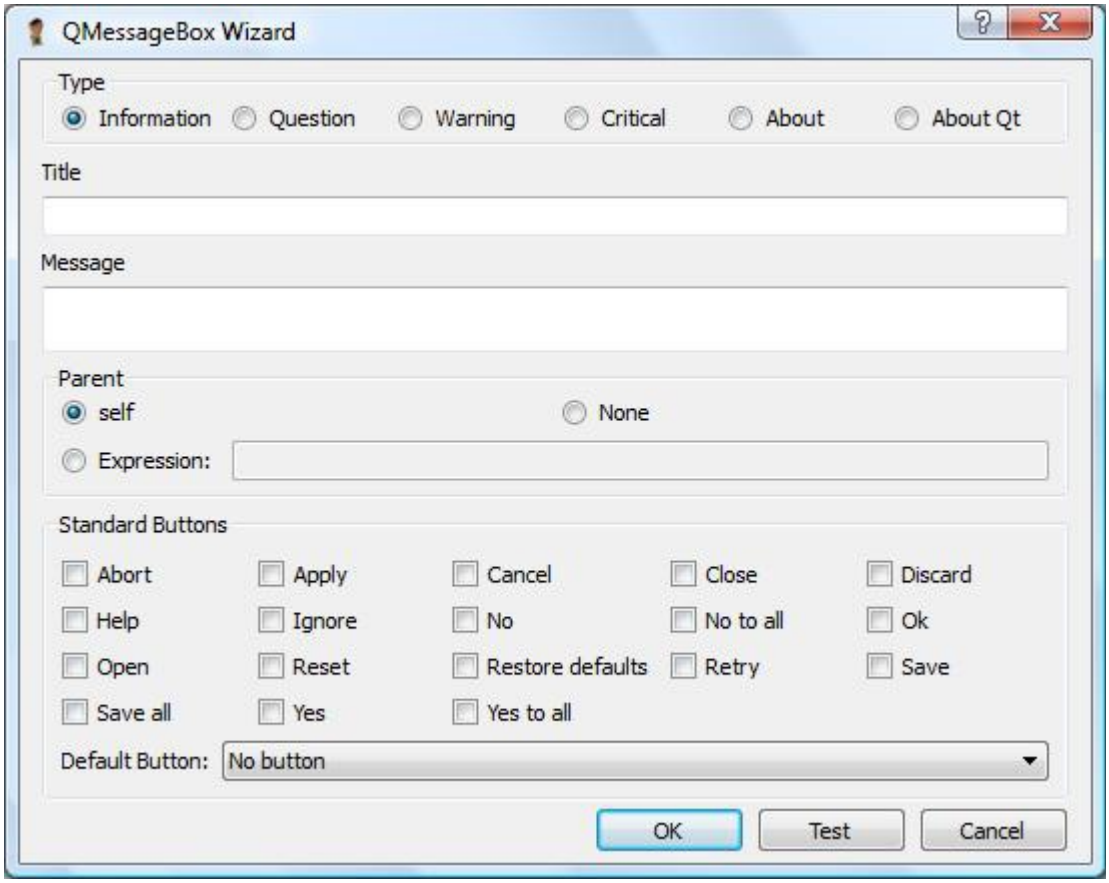
Extras > Wizards > **QFontDialog**

Extras > Wizards > **QInputDialog**

Extras > Wizards > **QMessageBox**

Extras > Wizards > **QRegExp**

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



Most of these software productivity tools are for PyQt[57] 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))
```

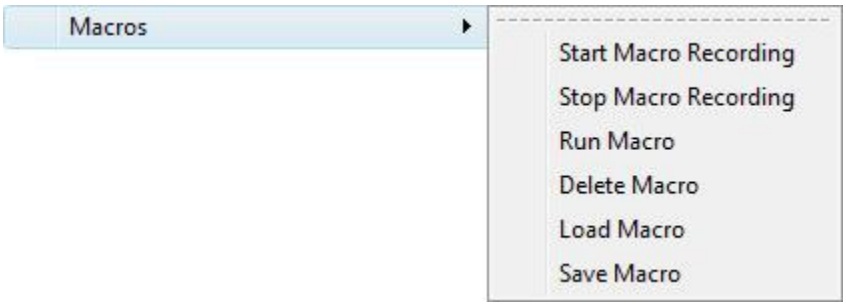
Viewpoint

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:* {4West}, *similar feature, more powerful*].



Remark

This is a functionality derived from the QScintilla toolkit[58], offering the possibility of recording source text fragments on-the-fly, 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.

Command List

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

Viewpoint

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

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

58 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 when 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*].

– –

Extras > Macros > **Run**

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

– –

Extras > Macros > **Delete**

To delete 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[59]. Then, to be `Run`, a `Saved` macro must be `Loaded` first.

– –

59 Default directory: <*User*>_eric5, File extension: "*.macro".

Extras > **Tools**

Designed to call a sub-menu of commands aimed at managing and using external Eric *Tools*.

Eric Tools are organized into three distinct *Tool Groups*: the two initial standard *Builtin* and *Plugin* Tools, as hereafter described [*see*], and then the *Custom* ones, possibly added subsequently.

Command List

`Select Tool Group  Configure Tool Groups...  Configure Current Tool Group...`[60]

– –

Extras > Tools > **Select Tool Group**

Designed to select so, then, to operate on one Eric Tool Group among the two standard "`Builtin`" and "`Plugin`" Groups or, possibly, the Custom Groups subsequently added by the end user [*see sub-command:* `Tools > Configure Tool Groups...`], if any.

These the visible effects on the selected Tool Group:

- Label name bold-faced;
- Related tool list enabled and shown at the bottom of the Tools sub-menu, after the `Tools > Configure Current Tool Group...` sub-command [*see*];
- Command `Tools > Configure Current Tool Group...` enabled, when selected a Custom Tool group [*see*].-

Builtin Tools is the only set initially available. For the actual presence of the other Plugin and Custom groups a specific user intervention is required. Actions actually enabled for each selected Tool Group are treated on subsequent sections.

60 <~> 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.

Command List

`Builtin Tools` `Plugin Tools` *`my Tool Group`*

Specifications:

`Builtin Tools` Standard Group of Eric Tools [*that is:* `Qt Designer`, ..., `Eric5 Web Browser`], as listed at next section: Extras > Tools (Select Tool Group = Builtin Tools) [*see*].

Viewpoint

We'd like to know something more about the reasons that led to the composition and the choice of this specific set of "`Builtin Tools`".

`Plugin Tools` Group pre-arranged for custom Tools possibly developed and configured via `Project > Packagers` menu command [*see*]. In a standard Eric installation this Group is initially empty.

`my Tool Group` Example of a possible Custom Tool Group, created with command: `Extras > Tools > Configure Tool Groups...` [*see*].

– –

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

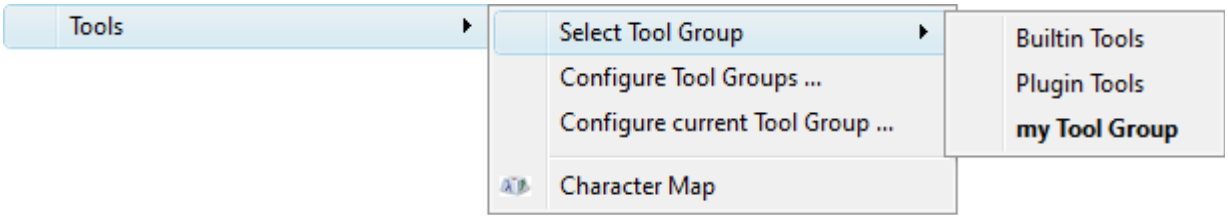
Designed to show a dialog box where to manage—create, delete, ...—the Custom Tool Groups.

That is the set of menu commands aimed at calling external toolkits which the end user can add besides to the two standard "`Builtin Tools`" and "`Plugin Tools`" Groups [*see*]. The "*`my Tool Group`*" as here shown is just offered as an example of how such a custom group can be created and added [*see also:* `Extras > Tools > Configure Current Tool Group...`].

As a consequence of such an addition the "`Select Tool Group`" sub-menu [*see*] will appear as here shown, with a custom "*`my Tool Group`*" item set and selectable just after the standard `Builtin` and `Plugin Tools` Groups.

– –

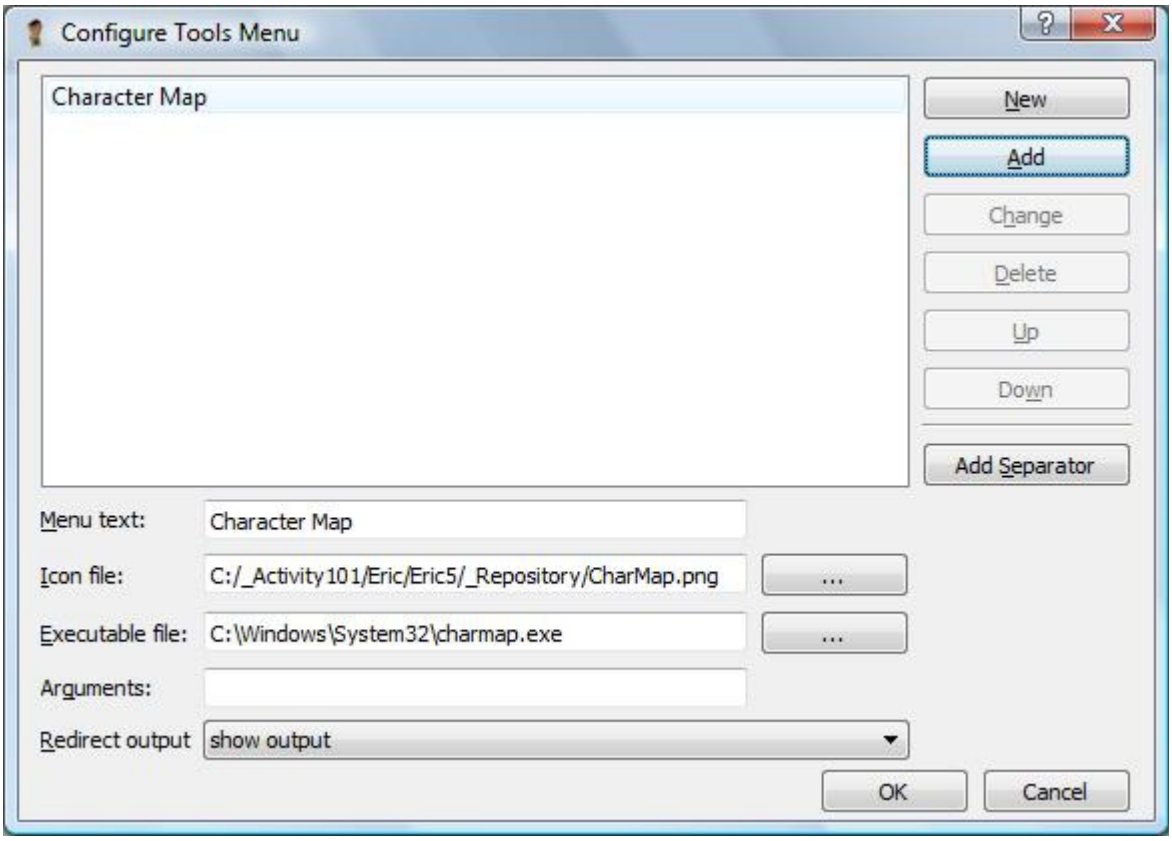
And this the appearance of a not-empty Custom Tools set, comprising a "`Character Map`" item as can be defined with the `Extras > Tools > Configure Current Tool Group...` command [*see*].



– –

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

Designed to show a control box where to configure the currently selected Custom Tool Group, such as the "*my Tool Group*" example as previously defined with command `Extras > Tools > Configure Tool Groups...` [*see*].

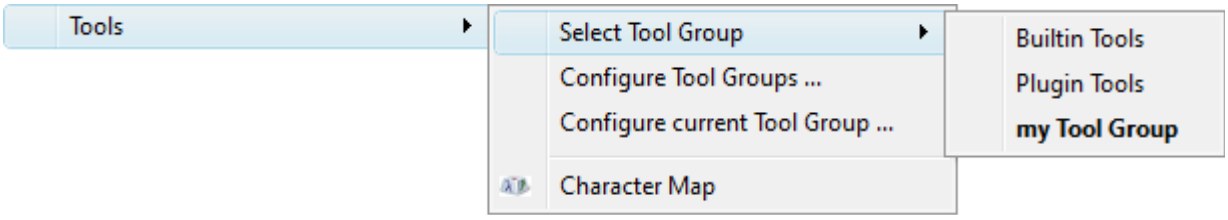


This is, primarily, a way for populating the set of external "`Executable file`" callable directly within Eric, with such an item as the "`charmap.exe`" in the example here shown.

Custom Tool Groups only can be here configured, not the standard "`Builtin`" or "`Plugin`" Tool Groups, which come already pre-configured. That's why this command results enabled only if there is at least one Custom Tool Group created via: `Extras > Tools > Configure Tool Groups...` [*see*].

Remark

The "`charmap.exe`" application here added as an example will become available to be called at the bottom of the `Tools > ...` sub-menu set, provided the related Custom Tool Group is selected [*see*].



Such `charmap.exe` system tool [*see*] 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 cannot be directly found on your keyboard, such as: `Alt 126`, for: "~", doesn't work well on the standard Eric text editor[61].

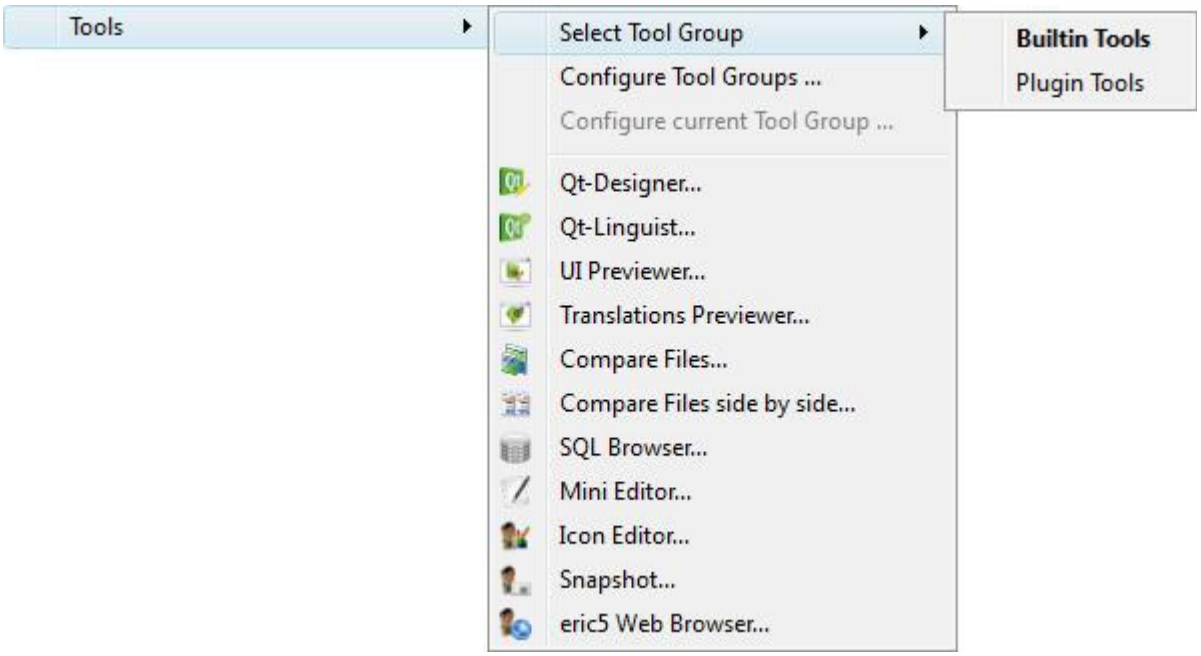
Remark

<!> Virtually any application available in the system in use can be here run this way.

– –

Extras > **Tools** (Select Tool Group = **Builtin Tools**)

"`Builtin Tools`" sub-menu, for executing specific utility programs as listed hereafter.



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

Command List

```
Qt-Designer...              Qt-Linguist...      UI Previewer...
Translations Previewer...   Compare Files...    Compare Files Side by Side...
SQL Browser...              Mini Editor...      Icon Editor...
Snapshot...                 Eric5 Web Browser...
```

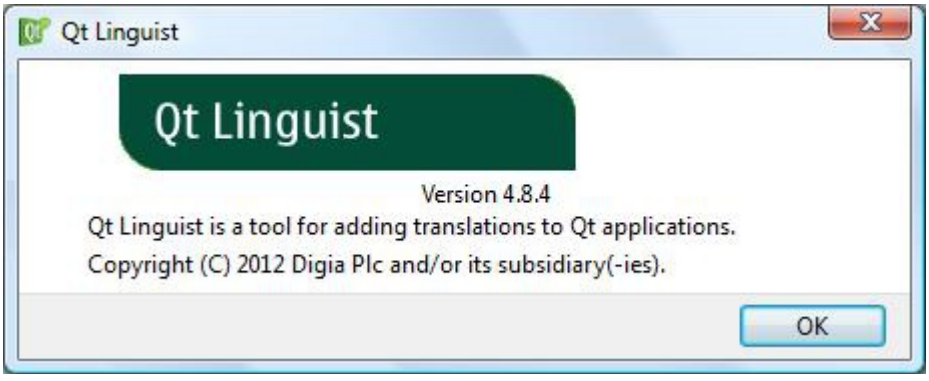
– –

Extras > Tools (Select Tool Group = Builtin Tools)> **Qt-Designer...**

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

– –

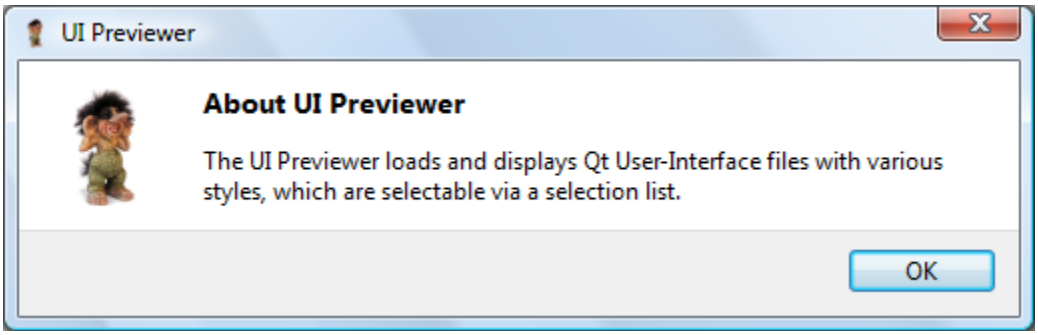
Extras > Tools (Select Tool Group = Builtin Tools)> **Qt-Linguist...**



A tool for adding translations to Qt applications.

– –

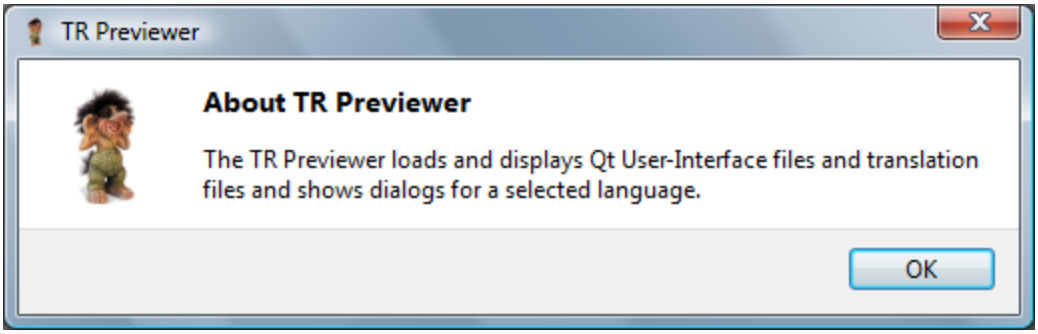
Extras > Tools (Select Tool Group = Builtin Tools)> **UI Previewer...**



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

– –

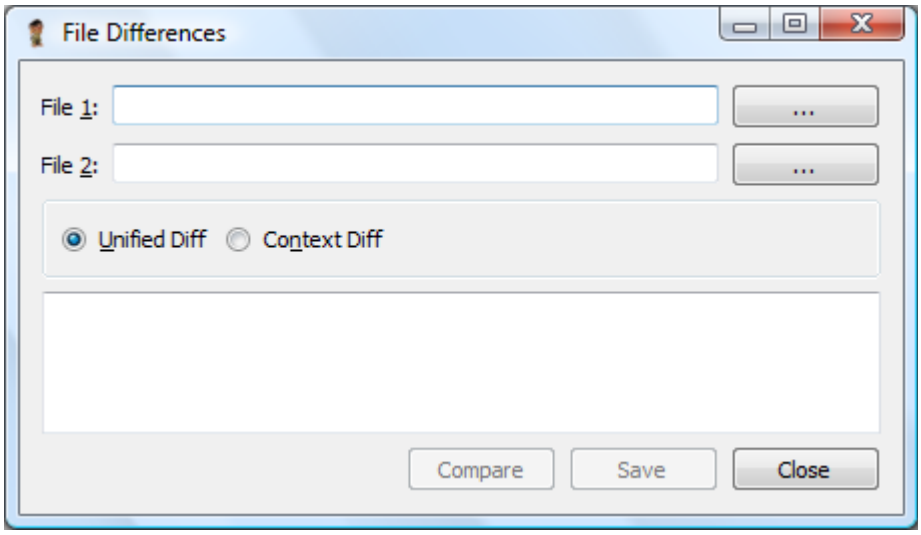
Extras > Tools (Select Tool Group = Builtin Tools)> **Translations Previewer...**



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

– –

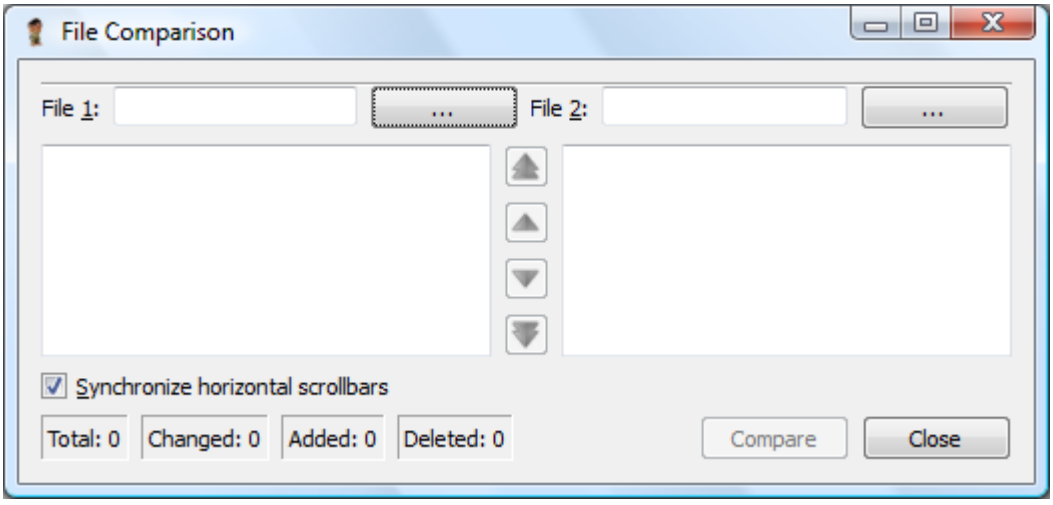
Extras > Tools (Select Tool Group = Builtin Tools)> **Compare Files...**



A tool for comparing text files, and showing differences.

– –

Extras > Tools (Select Tool Group = Builtin Tools)> **Compare Files Side by Side...**



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

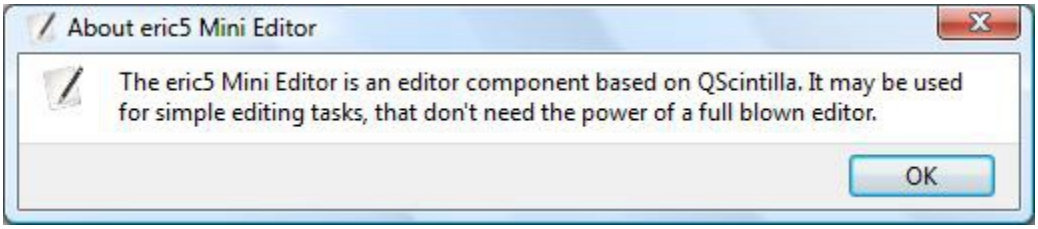
– –

Extras > Tools (Select Tool Group = Builtin Tools)> **SQL Browser...**

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

– –

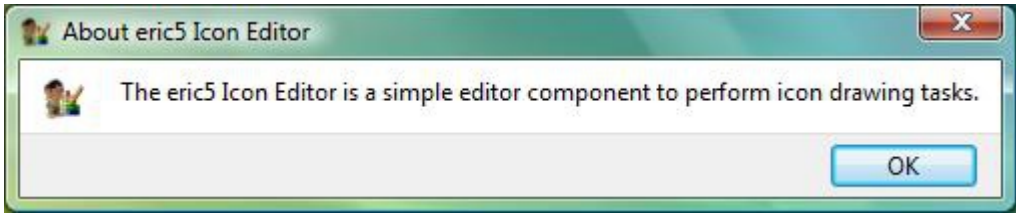
Extras > Tools (Select Tool Group = Builtin Tools)> **Mini Editor...**



A simple text editor, based on QScintilla.

– –

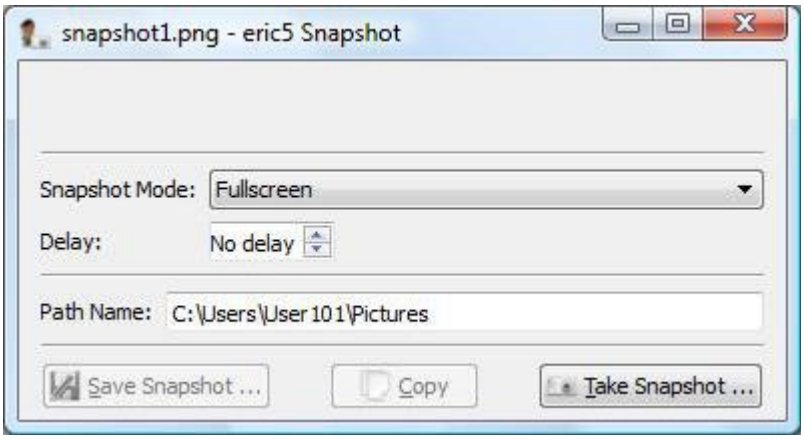
Extras > Tools (Select Tool Group = Builtin Tools)> **Icon Editor...**



A simple icon graphic editor.

– –

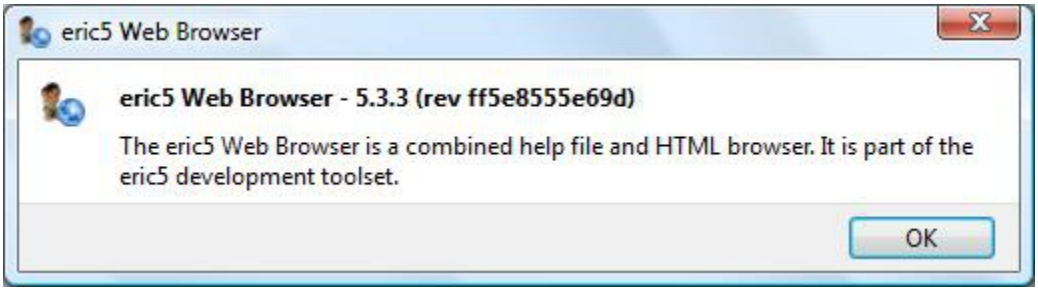
Extras > Tools (Select Tool Group = Builtin Tools)> **Snapshot...**



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

– –

Extras > Tools (Select Tool Group = Builtin Tools)> **Eric5 Web Browser...**



A combined help file and HTML browser [*cf.:* `Help > Helpviewer...`].

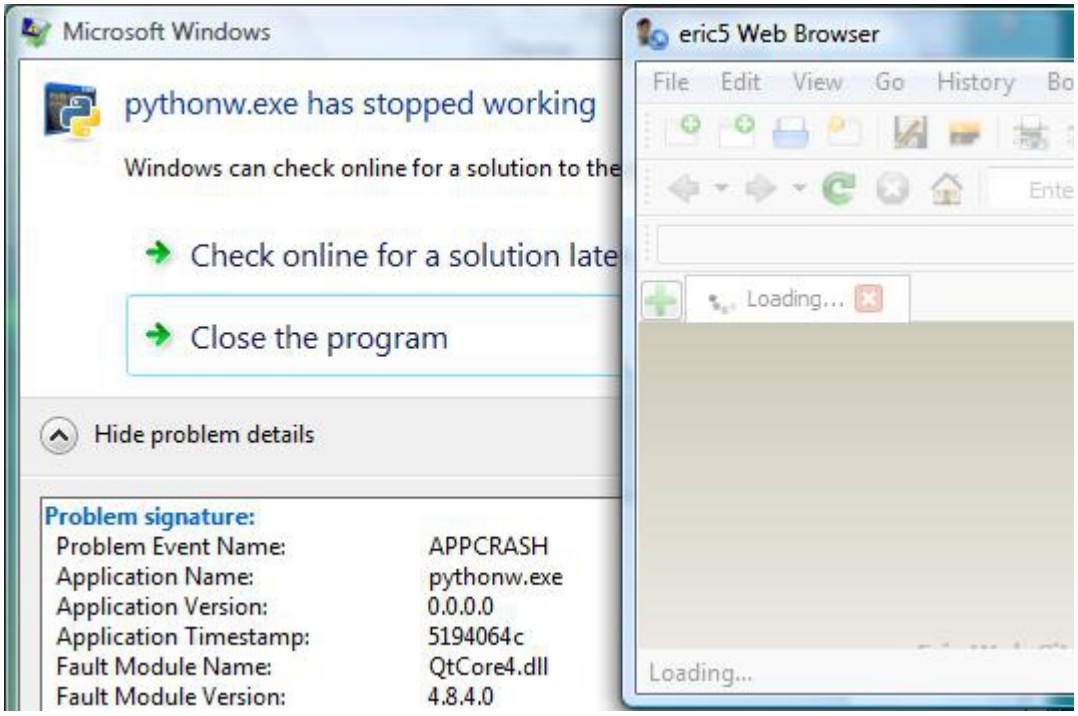
– –

Remark

<~> In case, with this Eric Web Browser just activated, you happen to get, unwonted, such alphabetic icons scattered all over the page:

You can easily get rid of them "Reloading" (`Ctrl+R`) the image. Such 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 alphabetic key to press so to activate the related link.

<~> Then, as another more serious glitch, if you happen to activate this same Eric Web Browser two consecutive times—for instance, as we happened to do, first with this `Extras > Tools (Select Tool Group = Builtin Tools)> Eric5 Web Browser...` and then with the equivalent `Help > Helpviewer...`, you can get Eric stuck on an endless "`Loading...`" / "`Failed to load`" loop.



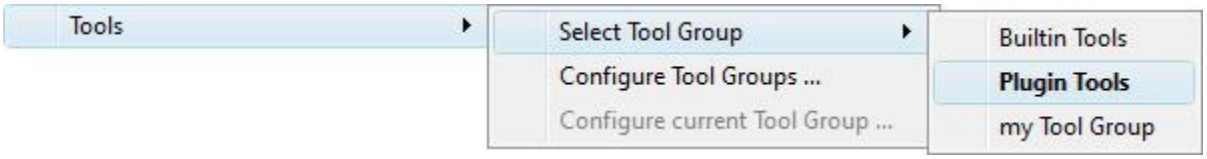
A—minor—issue now under investigation, probably due to a file that cannot be subsequently re-opened as beforehand not properly closed[62].

– –

62 Note that the "`Fault Module Name`" here declared is a "`QtCore4.dll`", that is not directly within Eric.

Extras > **Tools** (Select Tool Group = **Plugin Tools**)

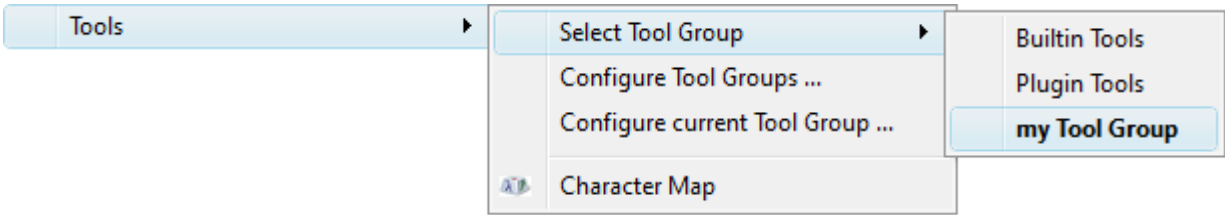
Designed to select the "`Plugin Tools`" Group, aimed at hosting custom Tools possibly developed and configured via `Project > Packagers` menu command [*see*].



In a standard Eric installation this Group is initially empty.

– –

Extras > **Tools** (Select Tool Group = ***my Tool Group***)



Command List

`Character Map`

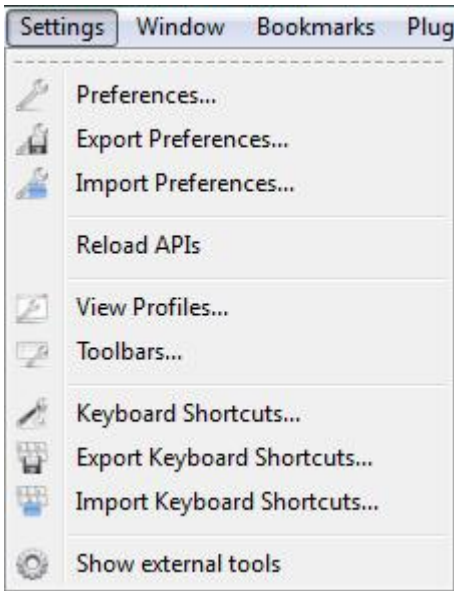
– –

Extras > Tools (Select Tool Group = *my Tool Group*)> **Character Map**

Here just as an example of a not-empty Custom Tool Group [*see:* `Extras > Tools > Configure Current Tool Group...`].

-<>-

Settings Command Menu



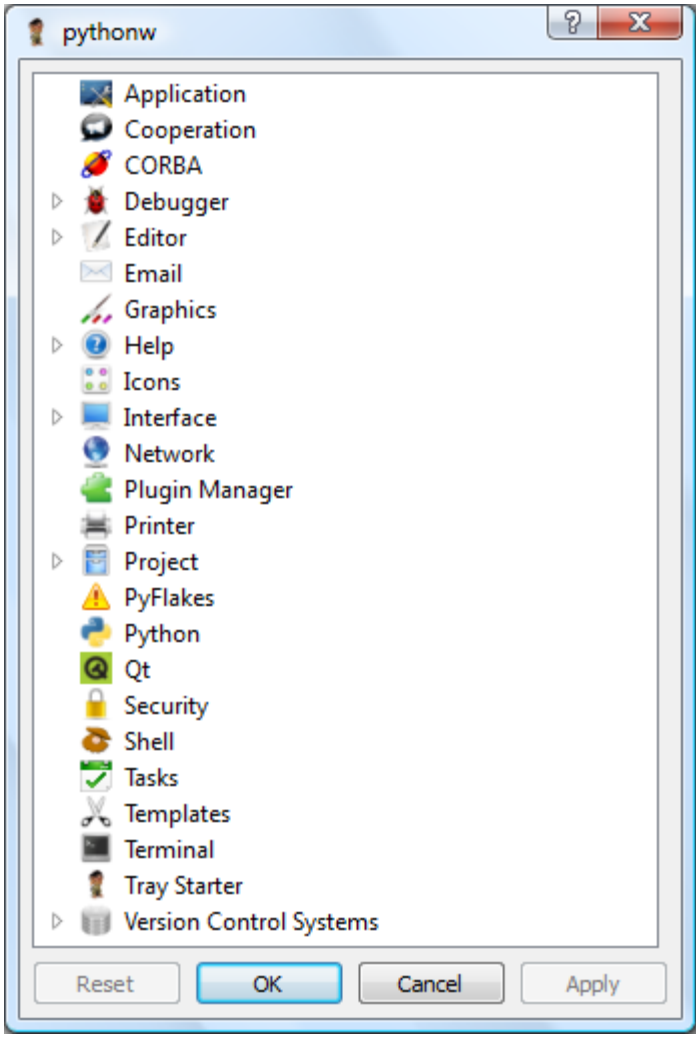
Command List

```
Preferences...          Export Preferences...          Import Preferences...
Reload APIs             View Profiles...               Toolbars...
Keyboard Shortcuts...   Export Keyboard Shortcuts...   Import Keyboard Shortcuts...
Show External Tools
```

– –

Settings > **Preferences...**

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



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

Viewpoint

<~> For such a rich Preferences tree, instead of a persistent full-expanded initial status, as it happens to be now, we'd suggest a memorizing expand / collapse mechanism, so to let the user conveniently concentrate his attention only where it's currently more useful.

– –

Settings > **Export Preferences...**

Settings > **Import Preferences...**

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

Remark

Such file appears to be organized into sections and keywords[63], not so difficult to recognize and interpret as a set of Eric preferences.

– –

Settings > **Reload APIs**

Here is all we know about this command:

- It will reload, that is, presumably, update, the API information used for call tips and auto-completion.
- Information contained in the configured API files.

Viewpoint

<。?。> Further investigation is here required, so to learn:

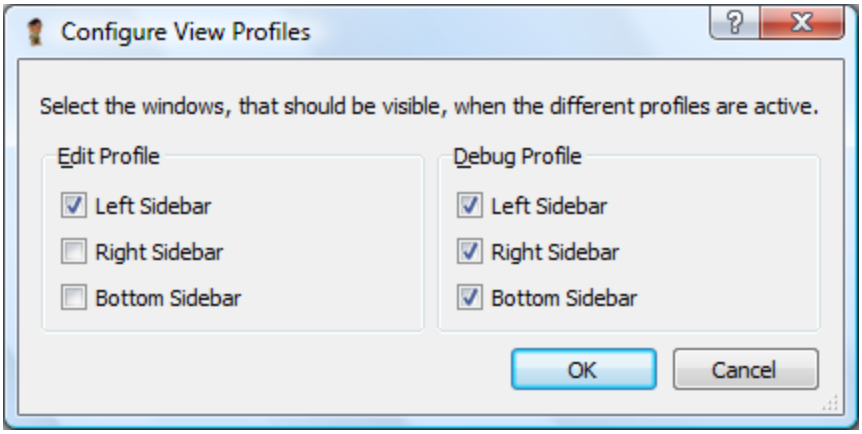
- When and why be it advisable to execute this command.
- Why is it required, instead of being automatic.
- Which are exactly such “configured API files”.
- How to possibly verify the effects on the involved call tips and auto-completion.

– –

63 Actually, it's a traditional Windows “`*.ini`”-like file.

Settings > **View Profiles...**

Command designed to show a control box where to define the two *View Profiles*[64] for the Edit and Debug operating modes, 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.

– –

Settings > **Toolbars...**

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

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

– –

Settings > **Keyboard Shortcuts...**

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

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

Set of commands designed to activate a configurator box to possibly personalize the keyboard shortcuts associated by default to menu commands. And then to possibly save / restore the related configuration file.

Remark

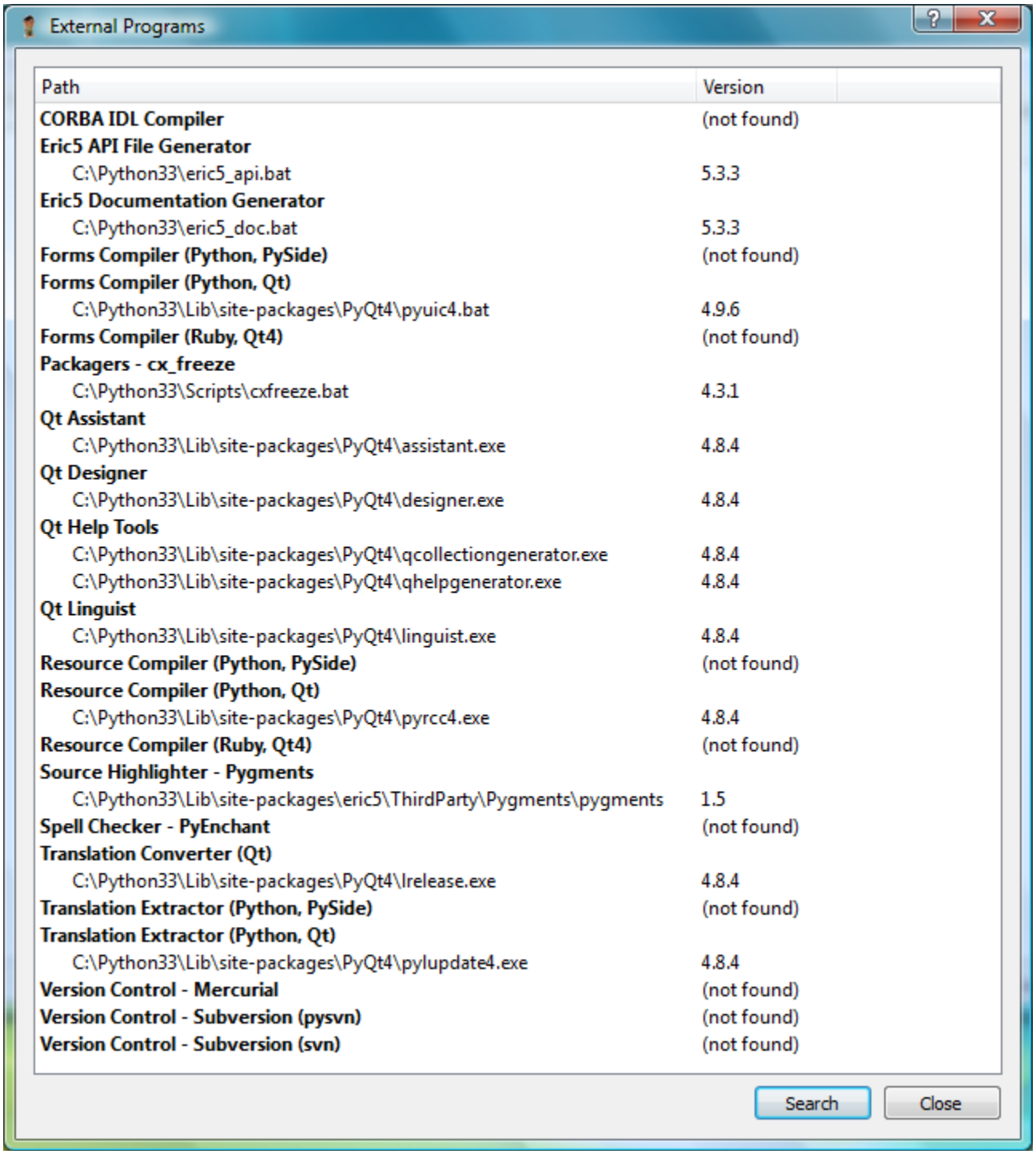
A configuration mechanism operating onto a specific XML file, with extension "`*.e4k`" and with a standard "`default.e4k`" version, located into the `Python3x\Lib\site-packages\eric5` directory [*see*].

Viewpoint

<~> Frankly we'd like to be more convinced about the validity of the exposure of such configuration mechanism to the normal user, that is not to Eric developers. Exposure that, we fear, could reveal more dangerous than useful. Anyhow a copy of the cited "`default.e4k`" file could be wisely used as a safeguard mechanism.

– –

Settings > **Show External Tools**



Designed to show the whole set of programs and tools that Eric, as an IDE, is designed to "Integrate" into one single "Developing Environment". Each item on the list with specified its actual attributes of path, version or "`(not found)`" flag.

Remark

For a distinction between mandatory and optional items see section: Setup and General Management [*in:* {6Trail}], and also command: `Help > Show Versions`.

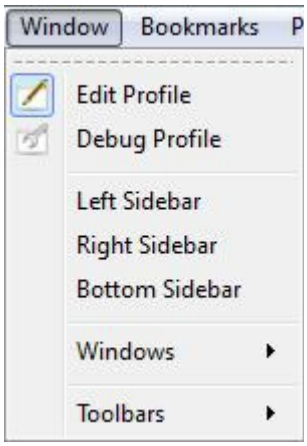
Viewpoint

<~> A useful[65] list, worth to be watched over and analyzed so to better evaluate the Eric's potential available to the user.

-<>-

65 <!> And rather impressive...

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 user selectable attributes.

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.

Different Auxiliary Forms will consequently be shown [*see also:* `Settings > View Profiles...`].

– –

Window > **Left Sidebar**

Window > **Right Sidebar**

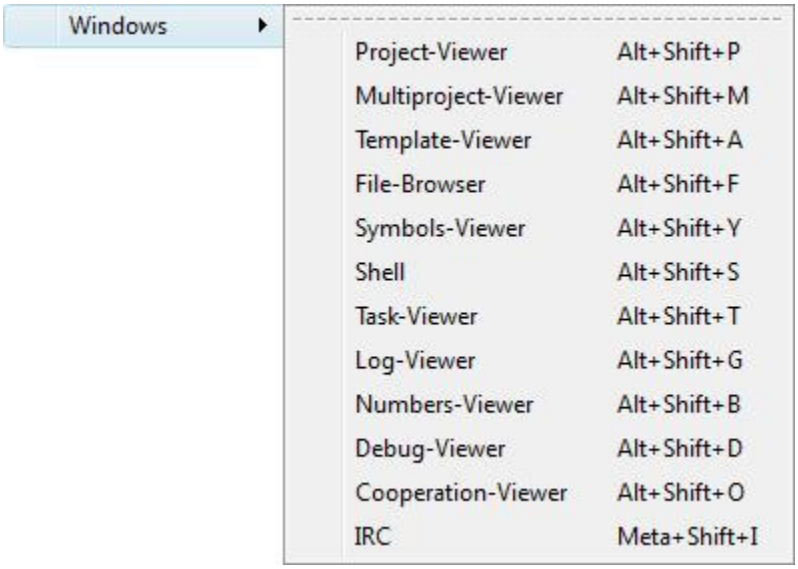
Window > **Bottom Sidebar**

Designed to show / hide the named Eric 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 select which of the listed control forms to show and select, on the hosting window pane. Same effect as a "`(^)`" click on the related tab, when in sight.

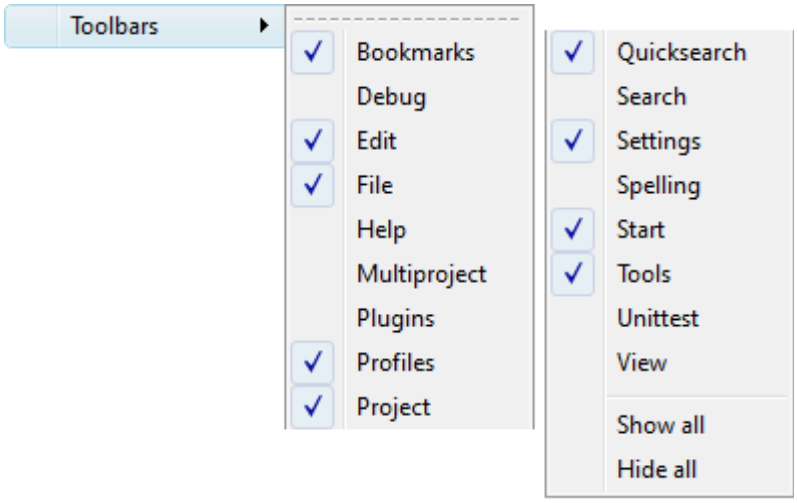


Related left / right / bottom hosting window pane will be shown, if not already in display [*see commands:* `Window > Left / Right / Bottom Sidebar`]. Detailed description of each one control form is hereafter offered on sections: Eric – South / West / East Side [*in:* {3South}, {4West}, {5East}].

– –

Window > **Toolbars**

Designed to show a sub-menu where to set the show / hide status for each and all Eric "Tools"—that is: `Bookmarks`, `Debug`, `...`, `View`—on the respective Tool-bars.



Here in display the initial default condition [*cf.:* Tool Bars, *last section here in:* {1North}].

-<>-

Bookmarks Command Menu

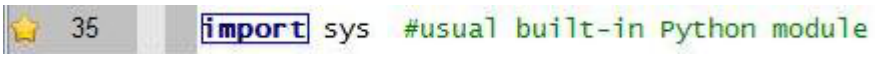
Command List

```
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 in 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

This action occurs anyhow at the program exit, with the command `Project > Session` [*see*] which is provided to possibly save / recall all bookmarks currently defined in a given Project..

– –

Bookmarks > **Bookmarks**

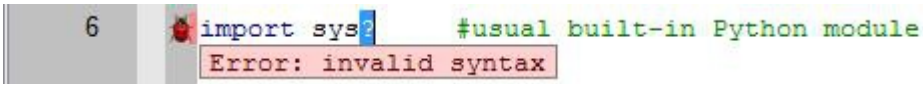
Designed to list all and, possibly, jump to one of the currently available source text bookmarks [*see also command:* `Bookmarks > Next / Previous Bookmark`].

– –

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 such first error detected. The offending line is "bug"-marked in the Notice column, which is the rightmost on the vertical Ruler Bar [*cf.:* Text Edit Central Pane, *in:* {2Central}].



This marking takes place at loading and saving of the related module, and also at typing of source text, after an idle time, configurable[66] [*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.

– –

Bookmarks > **Clear Syntax Errors**

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

– –

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

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 the presence of such related Notice icon ⚠, and the action of these commands remain unaffected.

– –

Bookmarks > Clear Warning Messages

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

Remark

Messages anyhow conveniently refreshed at any subsequent text editing 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`].

– –

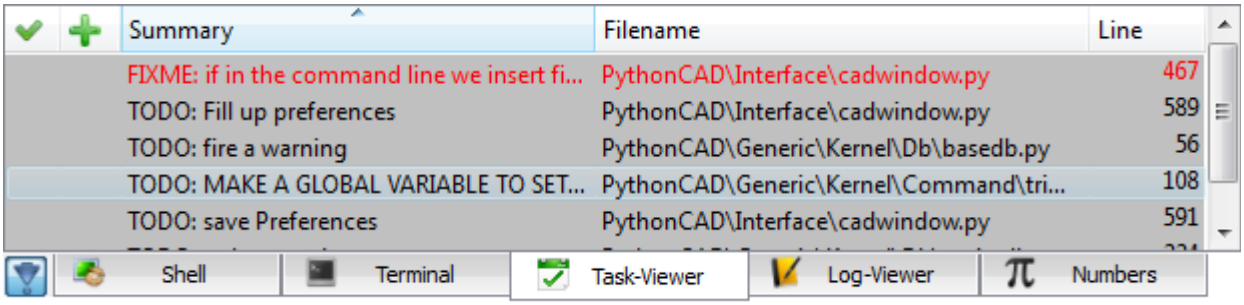
Bookmarks > Next Task

Bookmarks > Previous Task

Designed to jump to the next / previous line of the current source module[67] marked with such usual "*Global Task*"[68] 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, the rightmost on the vertical Ruler Bar [*see*]. Note that only the Task-markers typed ***exactly*** as in the related `Settings > Preferences... – Tasks, Tasks Markers` will be detected [*see*].

These Tasks can also be inspected and manually created and managed in the "`Task-Viewer`" of the tab Bottom Form [*see*], where, in case of a Project, they are all conveniently listed, independently from the opening state of the respective module.



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

68 "Global Task", so called to avoid any confusion with the different "Project Task" that can be created with the context command `Task (^) New Task...` [*see section:* B-Pane: Task-Viewer, *in:* {3South}].

Remark

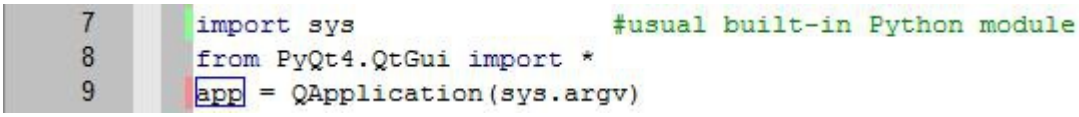
With Eric Projects, such Tasks are also recorded into a dedicated "`*.e4t`" file, located into the standard Project management sub-directory `<Project>\_eric5project`.

– –

Bookmarks > **Next Change**

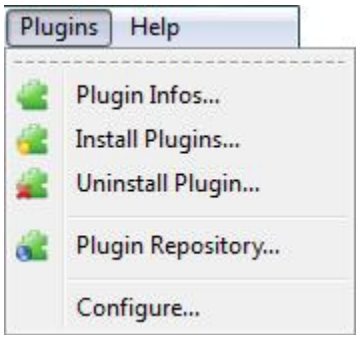
Bookmarks > **Previous Change**

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



-<>-

Plugins Command Menu



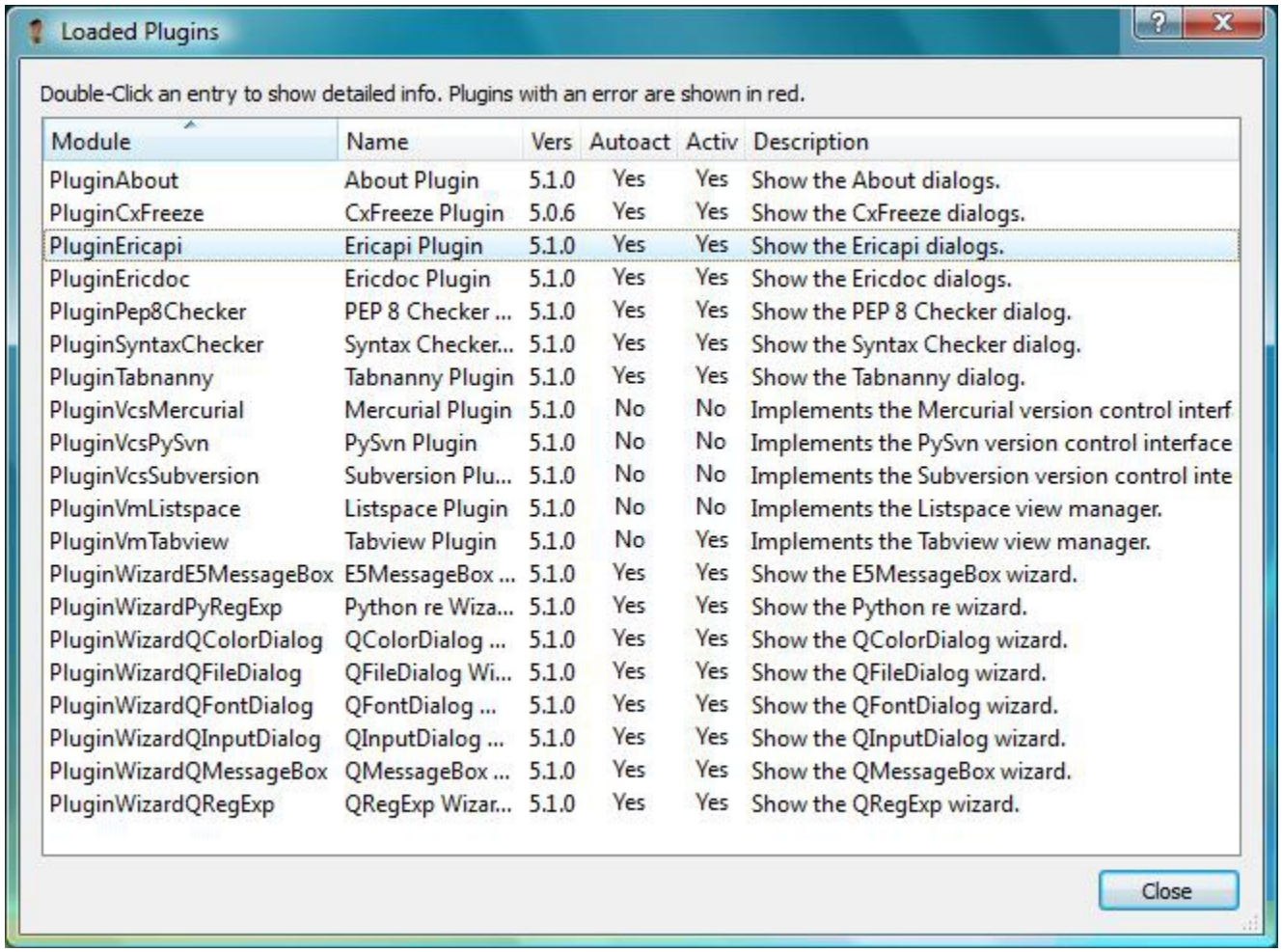
Command List

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

[*cf. command:* `Extras > Tools > Select Tool Group`].

– –

Plugins > **Plugin Infos...**



Designed to show a comprehensive "`Loaded Plugins`" table list form, where last eight "`Wizard`" Plugins—`E5MessageBox, ..., QRegExp`—clearly correspond to the same items shown in the `Extras > Wizards` sub-menu [*see*].

Remark

All items in this list are originally available as the result of a standard Eric setup action, to which then other ones can be possibly added by means of command `Plugins > Install Plugins...` [*see*]. For instance the "CxFreeze" item on the screenshot here in display is there thanks to a specific action of the `Plugins > Install` command, as hereafter describe in detail [*see also related* Remark].

And these are the Eric commands where the original and added Eric "`Loaded Plugins`", as here listed, can be found and used:

`About` `Help > About Eric5`

`CxFreeze` `Project > Packagers > Use cxfreeze` [*example of added plug-in*]

`Ericapi` `Project > Source Documentation > Generate API File (eric5_api)`

`Ericdoc` `Project > Source Documentation > Generate Documentation (eric5_doc)`

`Pep8Checker` `Project > Check > PEP 8 Compliance...`

`SyntaxChecker` `Project > Check > Syntax...`

`Tabnanny` `Project > Check > Indentations...`

`Vcs ...` `Project > Version Control`

`Vm ...` `Settings > Preference... – Interface, Viewmanager`

`E5MessageBox, ..., QRegExp`
Bottom eight Plugins in this list, as in command: `Extras > Wizards` [*see*].-

– –

Plugins > **Install Plugins...**

Plugins > **Uninstall Plugin...**

Designed to show a control form where to possibly install / uninstall Plugins downloaded from the Eric Web Repository [*see command:* `Plugins > Plugin Repository...`]. Installed Plugins will appear as new commands variously added to the standard Eric menu. For instance the `cx_Freeze` plugin add-on, as hereafter described, will appear as a "`Use cx_Freeze`" sub-command at the bottom of the `Project > Packagers` set of sub-commands [*see*].

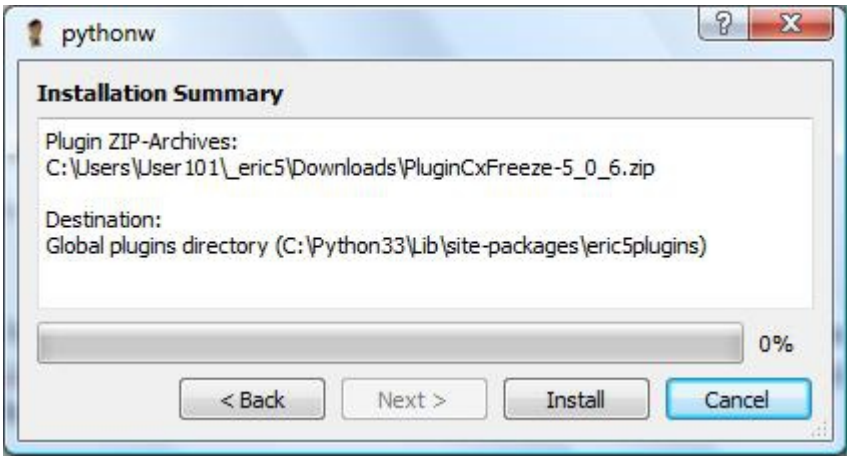
<!> The installation of these Plugins can imply considerable modifications on the Eric's user interface, such as: additional menus, new menu entries, and even new window panes. As a rule, in this Report we consider *exclusively the original user interface*, as shown in the Eric Window Map [*see in:* {Map}], possibly leaving the treatment of special Eric features and configurations to other dedicated documents [*see:* Scope of this Report, *in:* {0Lead}].

Remark

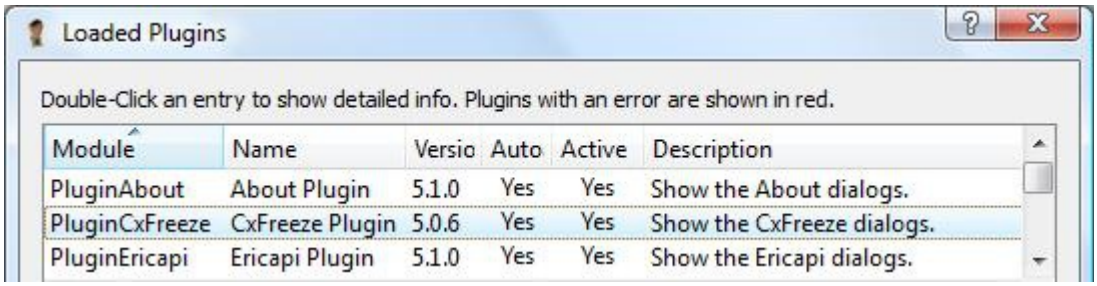
Just as a practical example of use, we have here tested this `Install` command procedure with the "`Packagers, CxFreeze`" Plugin, whose "`PluginCxFreeze-5_0_6.zip`" archive was previously

downloaded, and automatically stored into the directory "`<User>\_eric5\Downloads`" [*see:* `Plugin Repository...`]. Steps:

- Straightforward installation. Only question is about the usability scope, whether (current-)"`User`" or "`Global`".



- After such an `Install` action, the named Plugin could be spotted on the `Plugins > Plugin Infos...` table, where, if right-clicked, it will show a pop-up menu to: `Show Details, Activate, Deactivate` [*see*].



- As a result you'll then see a new "`Use cx_Freeze`" command ready to be used at the bottom of the `Project > Packagers` set of sub-commands [*see*].
- Anyhow note that this plugin is NOT an operative tool in itself. In fact it simply offers an interface front-end integrated into Eric for calling cx_Freeze[69], here assumed as known by the

69 cx_Freeze is a popular 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 Internet*].

user, and already installed and available into the current system.

Viewpoint

<~> We assume that such a set of Plugin tools, being external to the standard Eric, should possibly be tested and documented independently. That is, not in this Report [*see:* Scope of this Report, *in:* {0Lead}].

– –

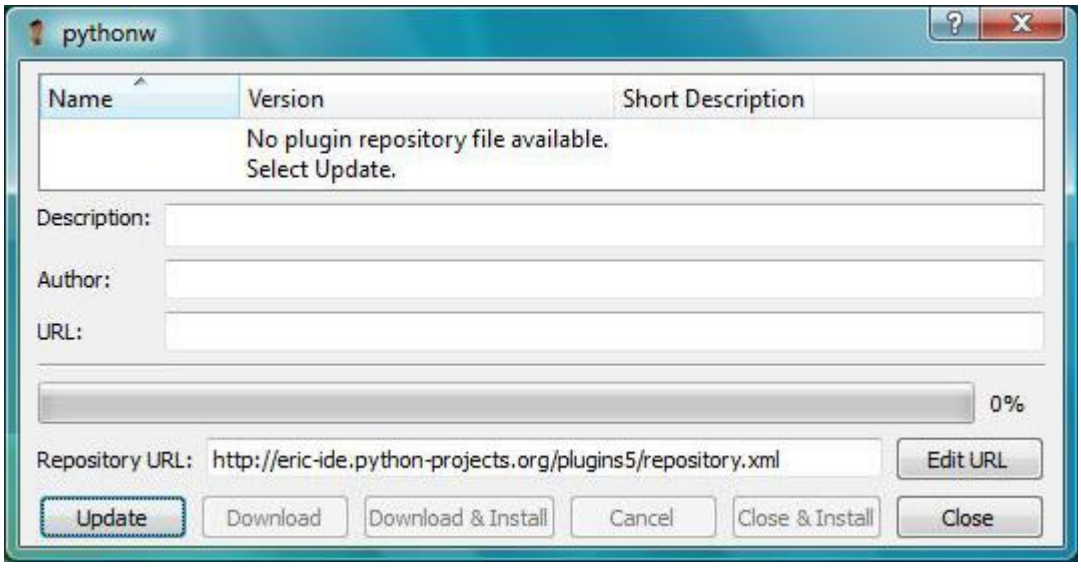
Plugins > **Plugin Repository...**

Designed to show a dialog box where to inspect, manage and, possibly, to update the local list of Eric Plugins currently available for downloading on the dedicated Eric's Plugins Web repository[70].

Update and download actions require an Internet connection active, all controls of this box are here assumed clear enough not to require any further explanations.

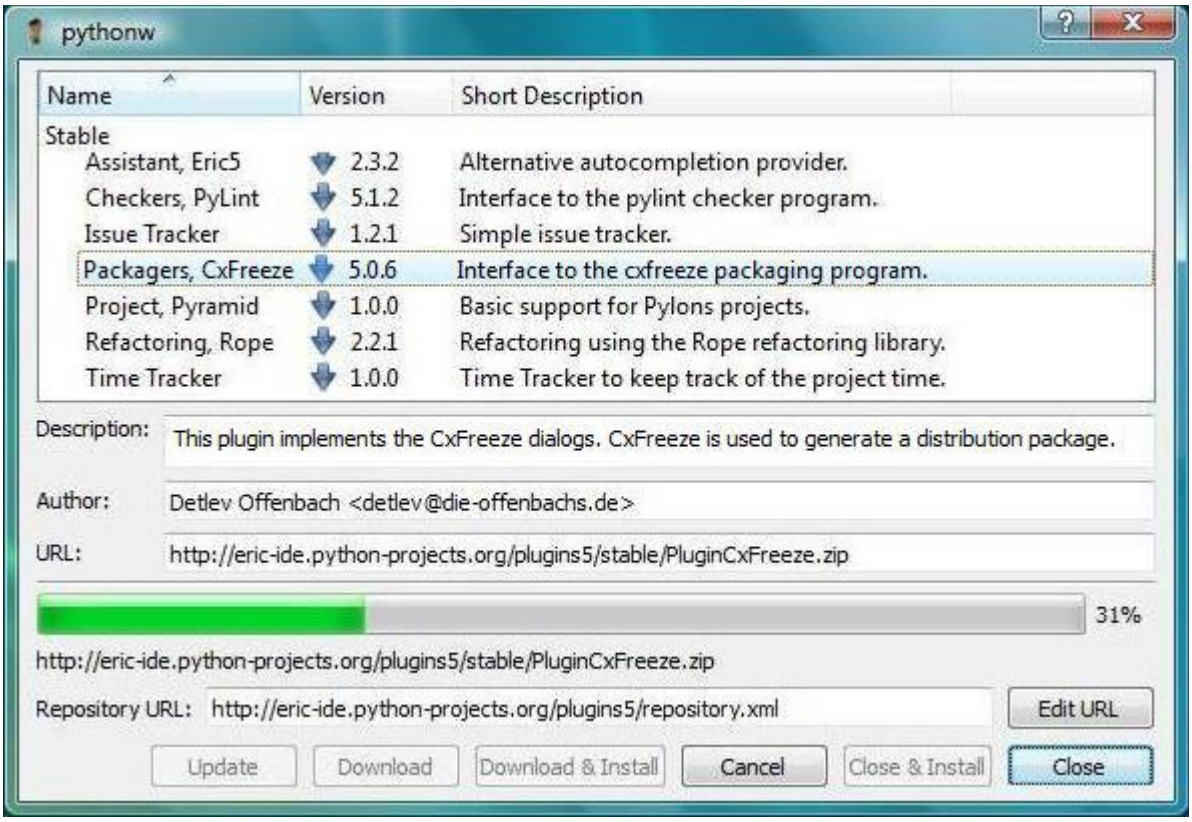
– –

This the initial aspect of the dialog box:



70 Current URL: http://eric-ide.python-projects.org/plugins5/repository.xml as here preset on the "Repository URL" field of this box.

And this the same dialog box as it appears during a "`Download`" action, after being "`Updated`":



Specifications:

- The top-left "`Stable`" label is to qualify this Repository, as opposed to another one provided for items "Under Development". And a "`Description`" is shown for the selected item.
- The downward-arrow symbol is for items "`Download`"-available, then missing if downloaded. And then re-established when a new version is available. Down-loaded items go to the current local Repository: `<User>\_eric5\Downloads` [*see also:* `Plugins > Configure...`].

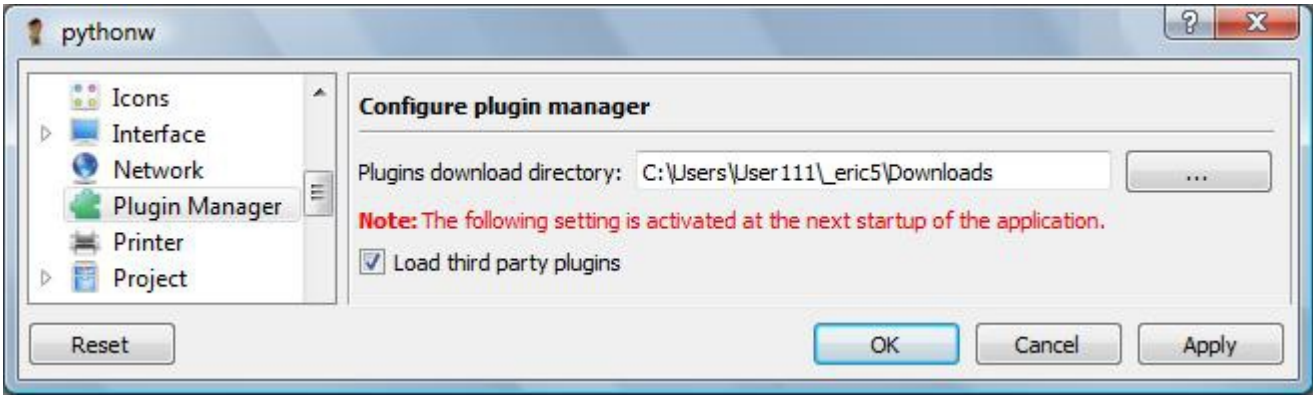
Remark

Note that this example is deliberately centered onto the `CxFreeze` package, because of its particular relevance [*see also:* `Plugins > Install Plugins...`].

– –

Plugins > **Configure...**

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



Where you can manage the destination directory where to possibly down-load the Eric Plugins, as for menu command `Plugins > Plugin Repository... > Update` [*see*].

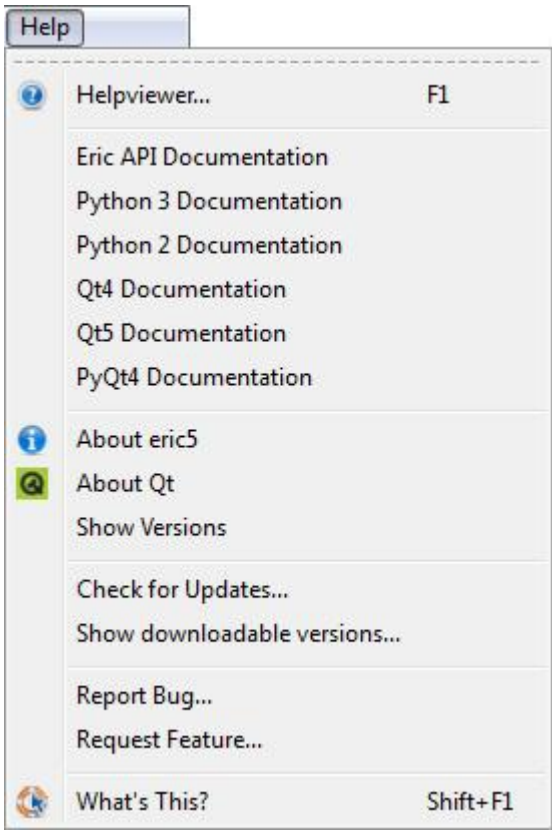
Remark

Here the check box caption "`third party`" should be actually red "`external`"[71], as it refers to the plugins not originally delivered together with Eric, but to those in the Eric Plugin Repository [*see command:* `Plugins > Plugin Repository...`], or any other similar sources possibly available.

-<>-

71 And so will be named with next Eric rev. 5.4.

Help Command Menu



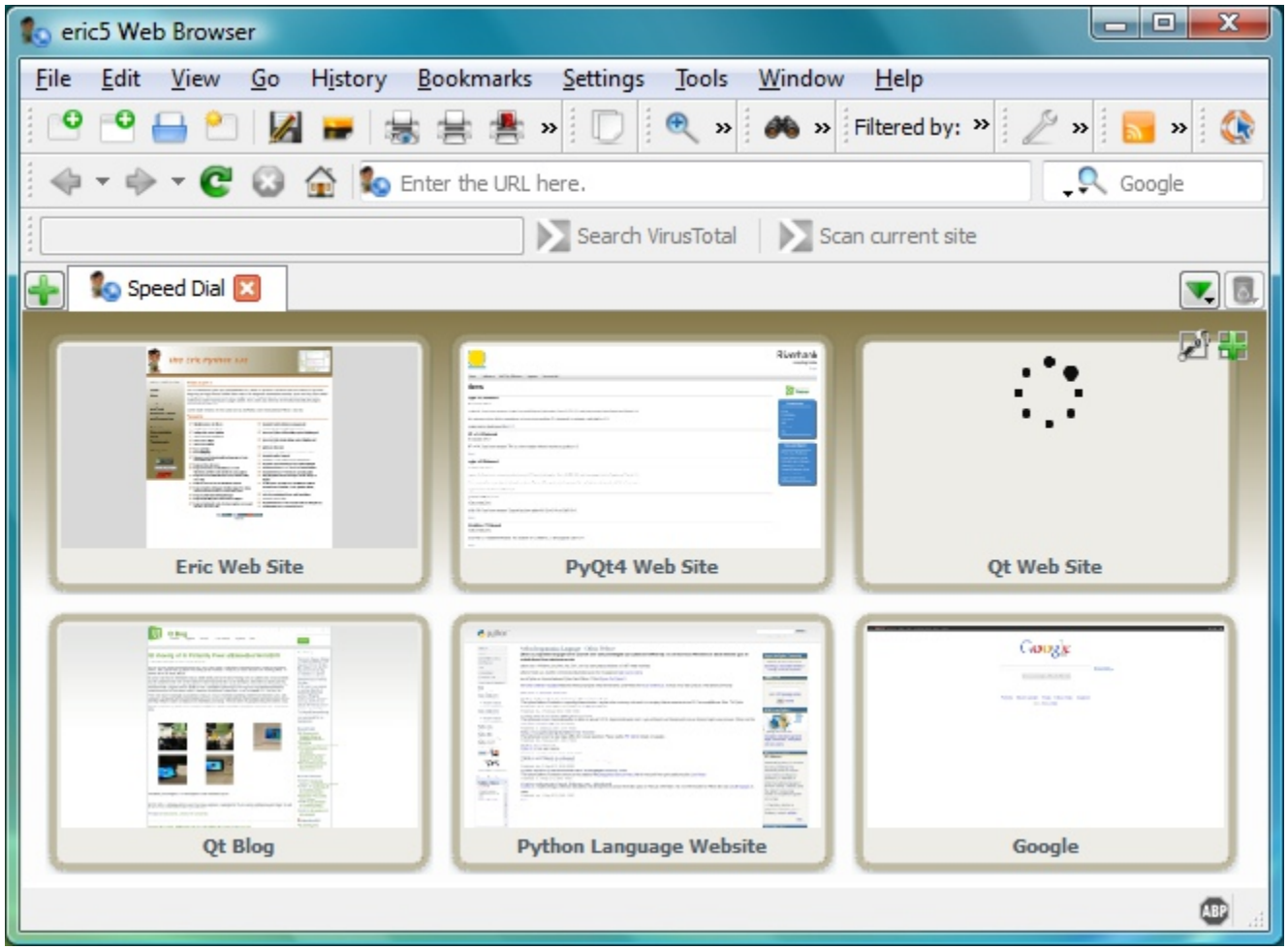
Command List

```
Helpviewer...              Eric API Documentation    Python 3 Documentation
Python 2 Documentation     Qt4 Documentation         Qt5 Documentation
PyQt4 Documentation        About Eric5               About Qt
Show Versions              Check for Updates...      Show Downloadable Versions...
Report Bug...              Request Feature...        What's This?
```

– –

Help > **Helpviewer...**

Designed to run the dedicated "Eric5 Web Browser" program, primarily aimed at browsing all Eric help material. Hereafter see its initial home page, with the "`Speed Dial`" thumbnails to the configured web sites.



It is basically a full-fledged Web Browser, even if not comparable with the major browsers currently in use, and which can be called explicitly, or automatically when inspecting the Help material here available [*see also:* `Settings > Preferences... – Help > Help Viewers, Eric Web Browser`].

Viewpoint

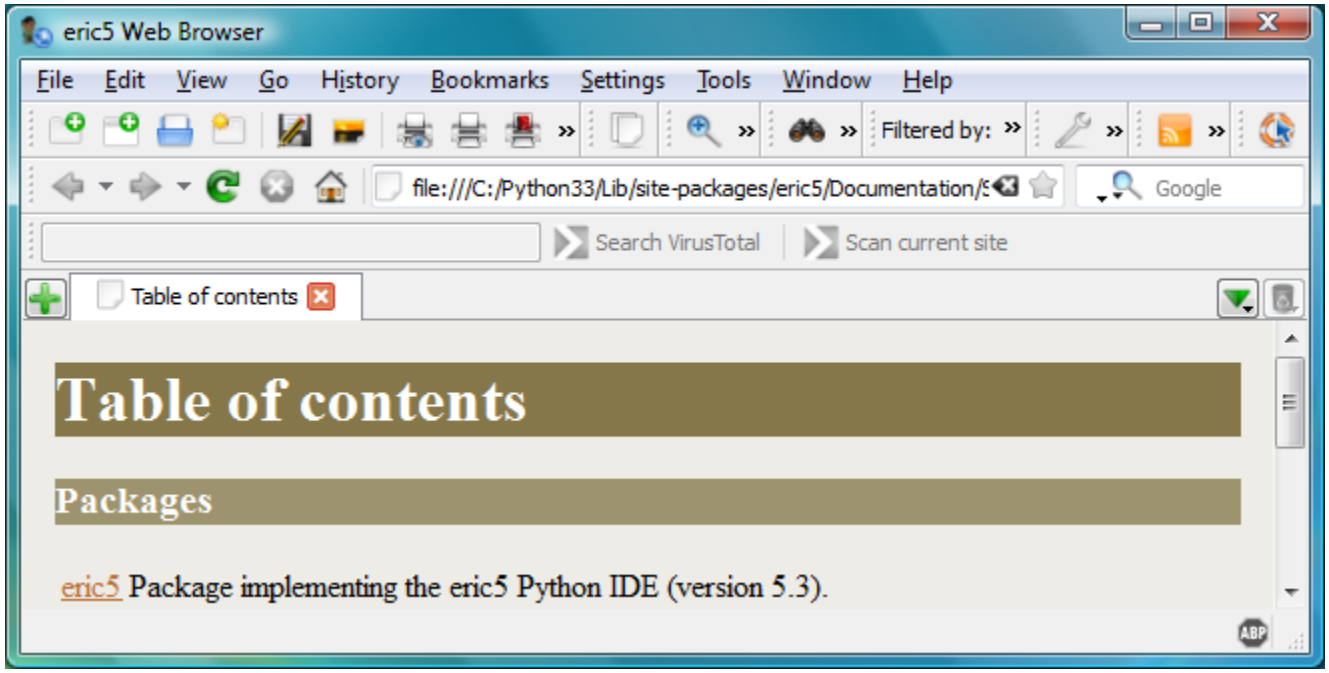
This is a self-standing subject of particular relevance, here just hinted at as assumed off the scope of the current Report [*see:* Scope of this Report, *in:* {0Lead}].

Detailed description of this application can be found under the title "Eric 4 Web Browser — Technical Report" in its practically identical Eric 4 version, on the Eric web site [*see:* Foreword, *What & Where in 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 this Eric application [*cf. command:* `Project > Source Documentation`].



Viewpoint

<~> Such Application Programming Interfaces (APIs) documentation is related to the very Eric development and, as such, it would be of interest more for an Eric's development team, than for the generality of its users. Who here would, obviously, appreciate much more a documentation of use[72], rather than of development.

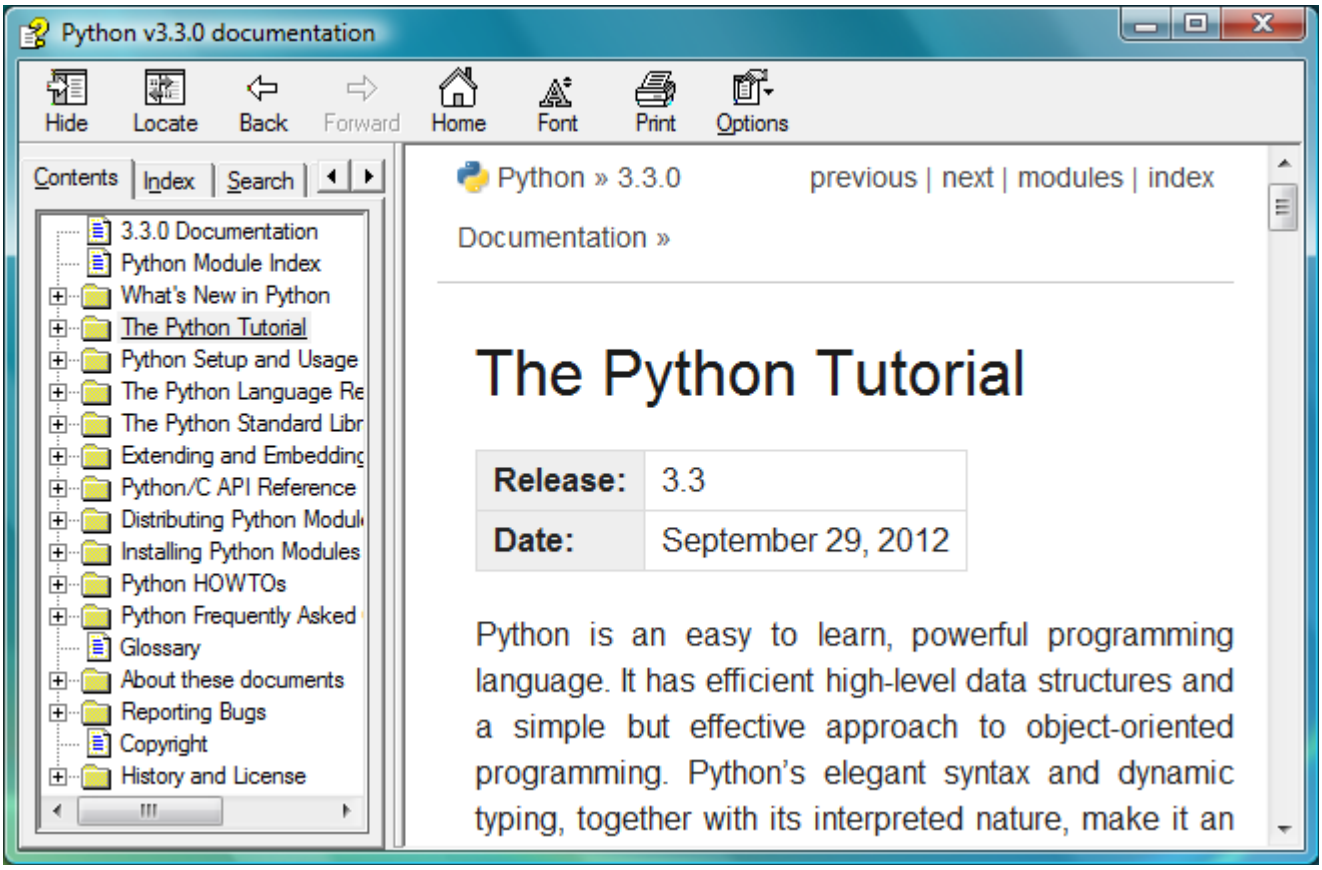
– –

72 As, for instance, an ordered 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[73], directly within Eric.



Remark

That is, provided that the appropriated paths, such as: `C:\Python33\Doc\python330.chm` and `C:\Python27\Doc\python271.chm`, had been assigned on: `Settings > Preferences... - Help > Help Documentation, Python Documentation` [*see also:* Eric Setup Completion, *in:* {6Trail}].

– –

73 Note, once more, that Eric 5 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. That intended as a source of information complementary to the `Help > PyQt4 Documentation` hereafter considered [*see*].

Remark

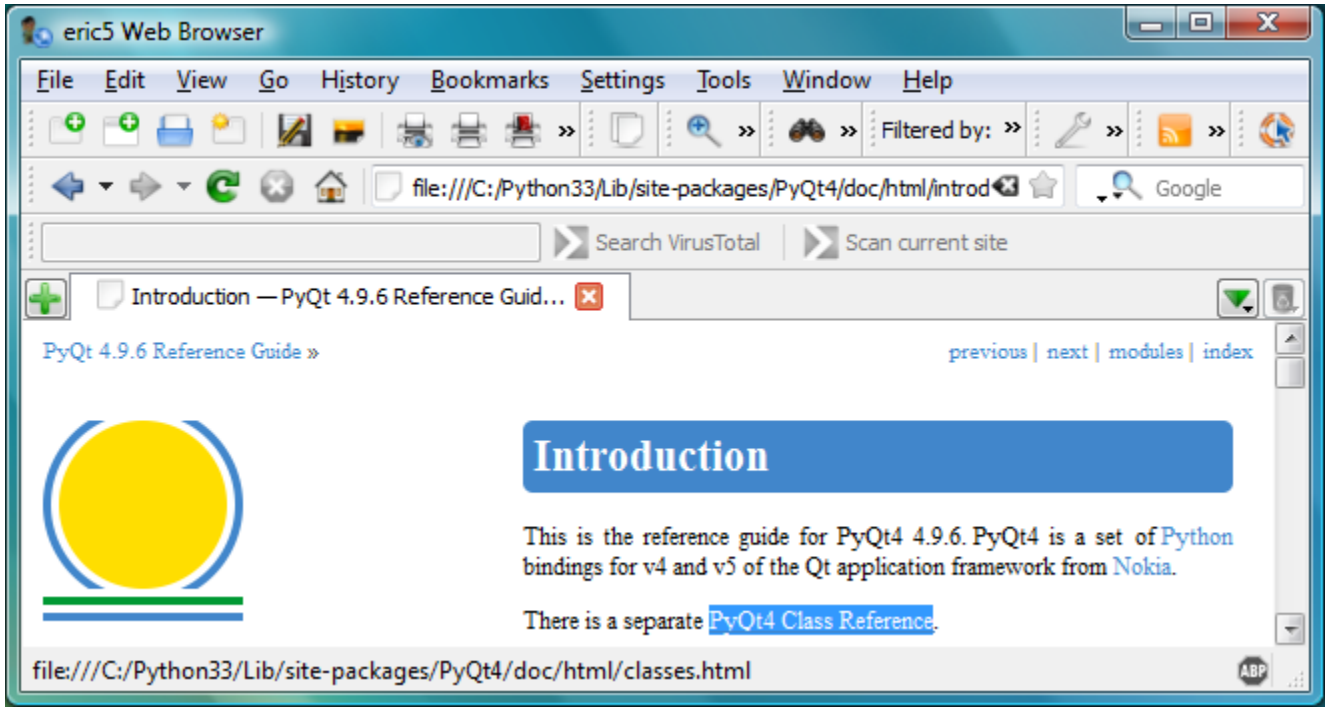
- Qt (for: "*cute*") is the cross-platform GUI toolkit at the origin of the of the PyQt Python binding, here considered being an Eric prerequisite [*see:* Prerequisites, *in:* {6Trail}]. It has got a documentation of its own[74], which is not available within the very PyQt package as here considered [*see: 2/2]* PyQt4, *in* {6Trail}].
- Further reference to this subject can be found on the Digia[75] (http://www.digia.com) and Qt Digia (http://qt.digia.com) web sites. Web references that can be possibly set as direct online links in these `Help` commands [*see:* `Settings > Preferences... – Help Documentation, Qt4 / Qt5 Documentation`].

– –

74 Whose role and utility is unknown to us. We mean: beyond the specific PyQt documentation.
75 Digia is the Finnish software company that has acquired Qt from Nokia in 2012.

Help > PyQt4 Documentation

Designed to gain access to the usual PyQt4 documentation[76], directly within Eric. More precisely, designed to open the Eric Web Browser on the "PyQt Reference Guide".



Where, in its Introduction page [*see*], you'll find also a useful link to the "PyQt Class Reference".

Remark

That is, provided that, PyQt4 toolkit installed, on: `Settings > Preferences... – Help > Help Documentation, PyQt4 Documentation`, such appropriated paths as: `C:\Python33\Lib\site-packages\PyQt4\doc\html\index.html`, had been assigned [*see:* Eric Setup Completion, *in* {6Trail}].

Viewpoint

This is a typical documentation for consultation only, rather unpractical for newcomers. For friendlier source of info see section Reference Book List [*in:* Appendix].

– –

76 Available as part of such standard "`PyQt GPL v4.9.6 for Python v3.3`" installation, in form of two PyQt Documentation items: "Reference Guide" and "Class Reference".

Help > **About Eric5**

Designed to show this product's identity card, comprising: official denomination, revision code, authors, acknowledgments and License (GNU – General Public License); along with two most appreciable e-mail references where users are invited to express their problems and wishes:

```
eric5-bugs@eric-ide.python-projects.org
eric5-featurerequest@eric-ide.python-projects.org
```

Viewpoint

Throughout this Report of the different ways, as here shown, to write this product's name, such as: ***Eric***, ***eric5***, ***ERIC***, for reasons of uniformity we have adopted this one:

Eric

with initial uppercase and, possibly, the final rev. number **5** only when it's appropriate to be so specific.

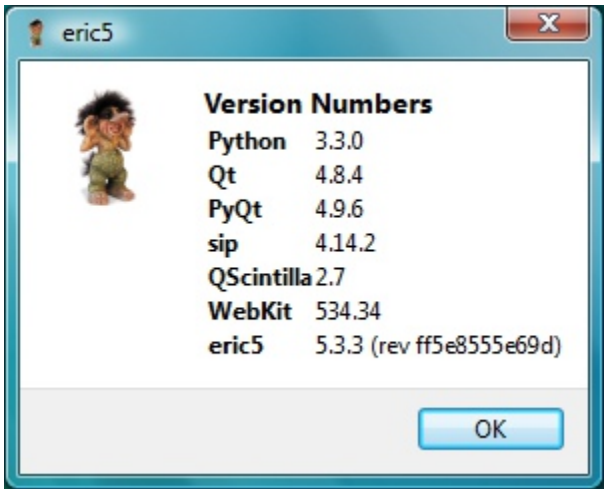
– –

Help > **About Qt**

Designed to show a copyright acknowledgment box for the Qt GUI toolkit, as here utilized. More precisely, this product is here utilized through its PyQt Python binding [*see:* Prerequisites, *in:* {6Trail}].

– –

Help > **Show Versions**

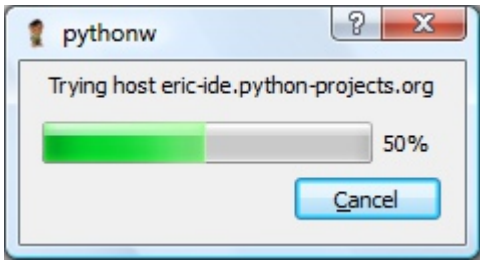


Designed to show a box listing all main items currently integrated into this unique IDE environment, along with their precise brand names and version codes [*cf.:* Prerequisites, *in:* {6Trail}].

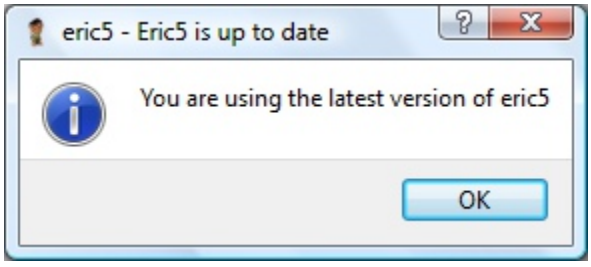
– –

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. To execute, this command requires an active Internet connection.



Here is a possible result:



For more information about all Eric versions available for down-loading see command: `Help > Show Downloadable Versions...`

Remark

The versions here considered are all relative to the same Eric 5 family, that is: not to the Eric 4, considered altogether as a different line of product[77].

– –

77 By the way, note that these two Eric 4 and 5 lines are compatible, in the sense that they can coexist within the same computer system.

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 command requires an active Internet connection.

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

Remark

Given a look at these links, they currently revealed as:

http://sourceforge.net/projects/eric-ide/files/eric5/stable/5.3.3
http://sourceforge.net/projects/eric-ide/files/eric5/unstable/5.3-20130106

– –

Help > Report Bug...

Help > Request Feature...

Two commands aimed at granting the users a channel of dialog with the Eric producer.

Viewpoint

A highly appreciable attitude, confirmed also by the e-references of dialog offered in an even simpler[78] and direct way, with the `Help > About Eric5` command [*see*].

– –

78 Simpler, as it doesn't require such `Configure Email` settings as the "`Outgoing mail server (SMTP)`" and the "`Outgoing mail server port`" into the `Setting > Preferences` dialog box, as here required [*see*].

Help > **What's This?**

Designed to show a short description on a pop-up help box, just when pointing at a 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 to "*`Perform unit tests`*", which is an assertion so obvious to be 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 to clarify something like: "*`the standard Python unittest module`*", to be of real help to the poor Mr. User. Who, this way, would be pointed to the right direction where to go so to learn more, if he wished to.

-<>-

Tool Bars

Default Tool Bar, on the "North Side" of the Eric Application Window, below the Main Menu Bar, as it appears in `Window > Edit Profile` mode [*see*]:

And at the right of the window, in `Window > Debug Profile` mode [*see*]:

Set of icon-tools designed to constitute an efficiency aid for calling the most frequently used menu commands. Role of each icon is hinted by a tool tip, such as the "`Open (Ctrl+O)`" shown in this figure, reporting both the corresponding menu command's name[79] and its short-cut.

Functional description of each icon's command can be found on the dedicated section Menu Bar, and Commands [*at top here in:* {1North}]. Choice and presence of the set of tool-icons can be managed and customized with dedicated commands: `Window > Toolbars` and `Settings > Toolbars...` [*see*].

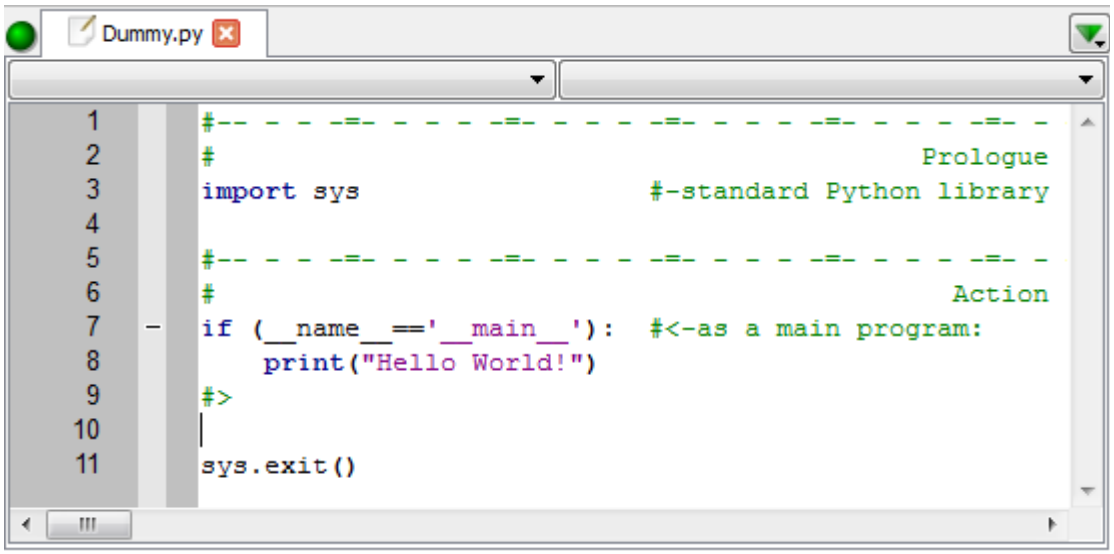
– = –

79 So that you may refer to the related Report section for a complete functional description of the named command.

{2Central} Eric Window – *Central Park*

Text Edit Central Pane

A stable pane—that is: not optional [*see next sections: "*Auxiliary ... Pane"]—positioned at the center of the Eric window's work area, aimed at hosting a tabbed set of 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*].

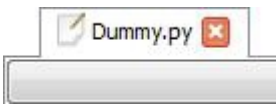


To this *Text Form* is associated a powerful *Text Editor*, so intuitive that it doesn't require any explanation, and also some context menus as described in detail in the subsequent two sections [*see*].

– –

Control List

Green pin marker, for the currently active Tab; red colored, for the not-active Tabs. Particularly relevant for split views [*see command:* `View > Split View`].

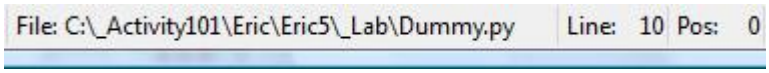


Tab control, for the Text Form selection, with a "**x**"-`Close` button.

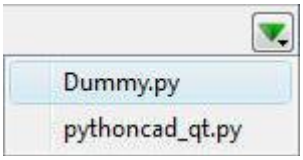
Remark

Note that on the Information Bar, at the bottom of the Eric Window, you'll see the path of the currently displayed file, and also the line–character position of the text insertion point.

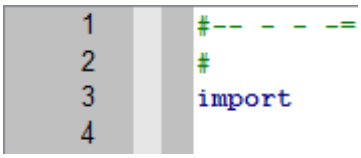
E.g.:



– –

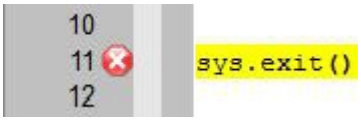


Navigation list, at the top-right of the text edit pane.
Click to show

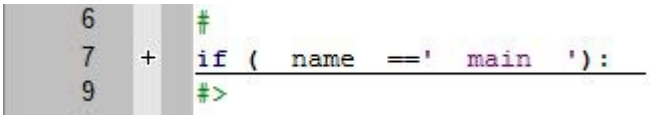


Vertical Ruler bar, to show the text line numbers and, possibly, the toggle control marks of, in left-right order: Bookmark, Breakpoint, Expand / Collapse Fold, and Notice icon [*see hereafter*]

Bookmark "star", with associated context menu [*see next section*]



Breakpoint "cross", with associated context menu [*see next section*]

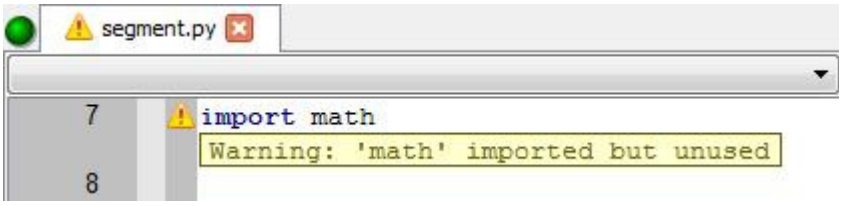


Expand / Collapse “fold” (“+/-”) source block.

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 [*indeed: default, as usual in this* Report].

– –



Notice icon, and message. E.g.: a “!”-Warning.

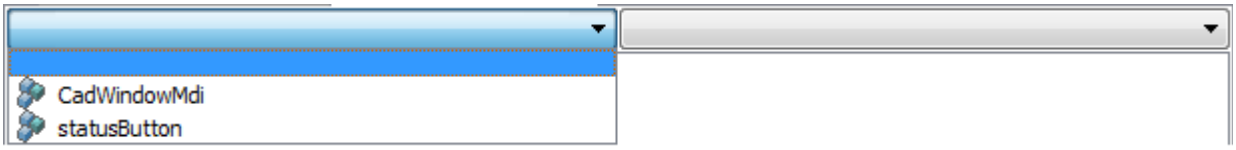
– –

Drop-down List of Objects

A pair of drop-down lists, designed to see and navigate amongst the objects of the current module.

The first list, at left,

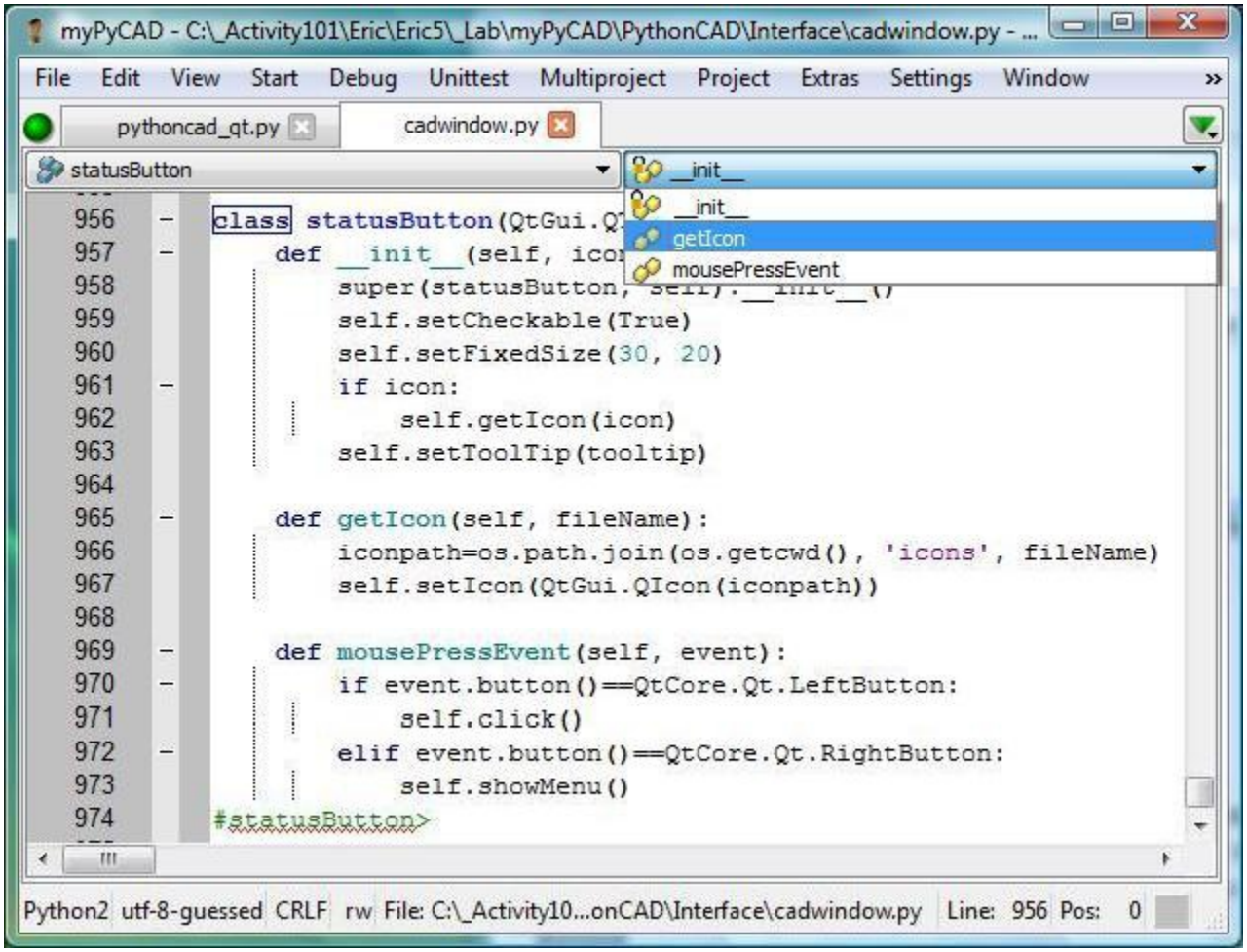
is for the classes and functions.



– –

The second, at right,

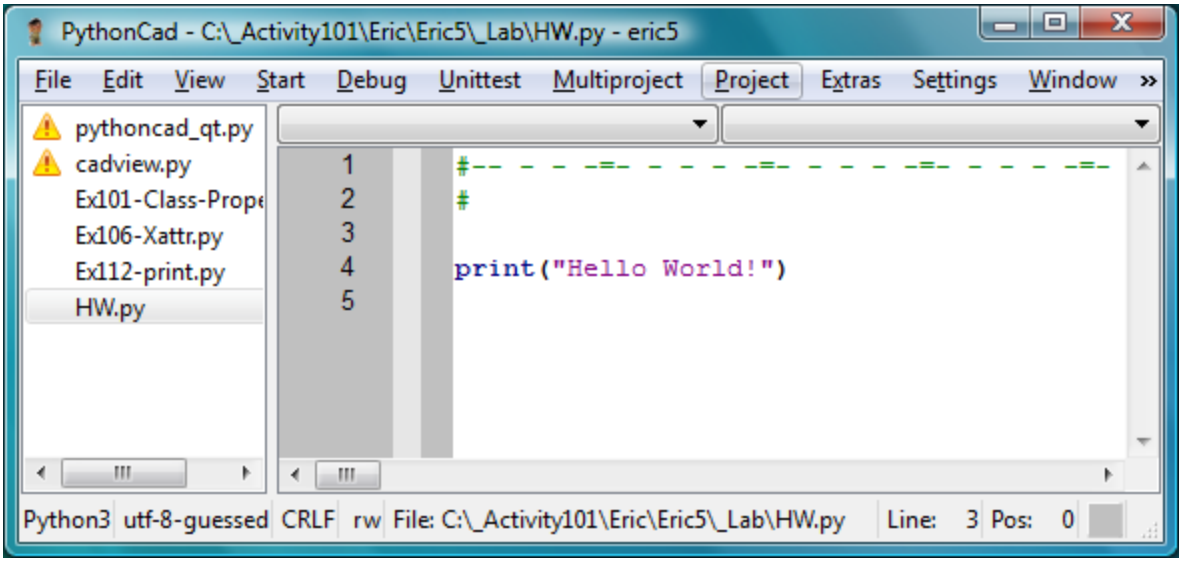
is for the methods and attributes of the class possibly selected on the first list, at left [*see following "*`statusButton`*" example*].



– –

Alternative Text “List View” Form

A text “List View” form can be activated as an alternative to the text tabbed form shown so far.



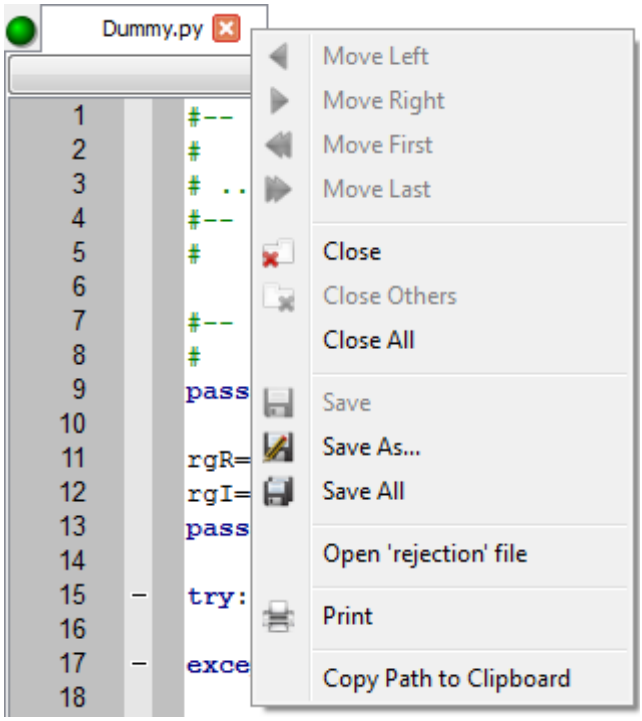
To that purpose select `Window view: Listspace` on the `Configure viewmanager` control form, via command `Settings > Preference... – Interface > Viewmanager` [*see*]. That is, as an alternative to the initial `Tabbed View` default condition. Such new condition will be then enabled at the next application 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.

-<>-

Tab (^) Menu



Remark

<!> This is the first one of the many, many context pop-up menus that enrich Eric's usability, as hereafter described. To activate:

- Point the mouse at the Tab, then right-click. Action represented throughout this Report with a symbolic: "(^)"
- Alternative action, valid only on the source text: (Shift+F10), or Application key.

Command List

```
Move Left     Move Right    Move First    Move Last
Close         Close Others  Close All                    [see menu: File >]
Save          Save As...    Save All                     [see menu: File >]
Open 'Rejection' File
Print                                                    [see menu: File >]
Copy Path to Clipboard
```

– –

Remark

<!> Eric context-menus comprise both original context-dependent commands and also some of the main menu commands, there duplicated just for reason of convenience.

As a rule, only the original context-dependent commands will be hereafter described, whereas, for duplicated commands, reference of the corresponding main menu will be offered, with no description duplicated.

– –

Tab (^) **Move Left**

Tab (^) **Move Right**

Tab (^) **Move First**

Tab (^) **Move Last**

Designed to move the currently active Text Form to the named position, within the set of tabbed forms. Same action can be done through the drag-and-drop of a Tab.

– –

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.

– –

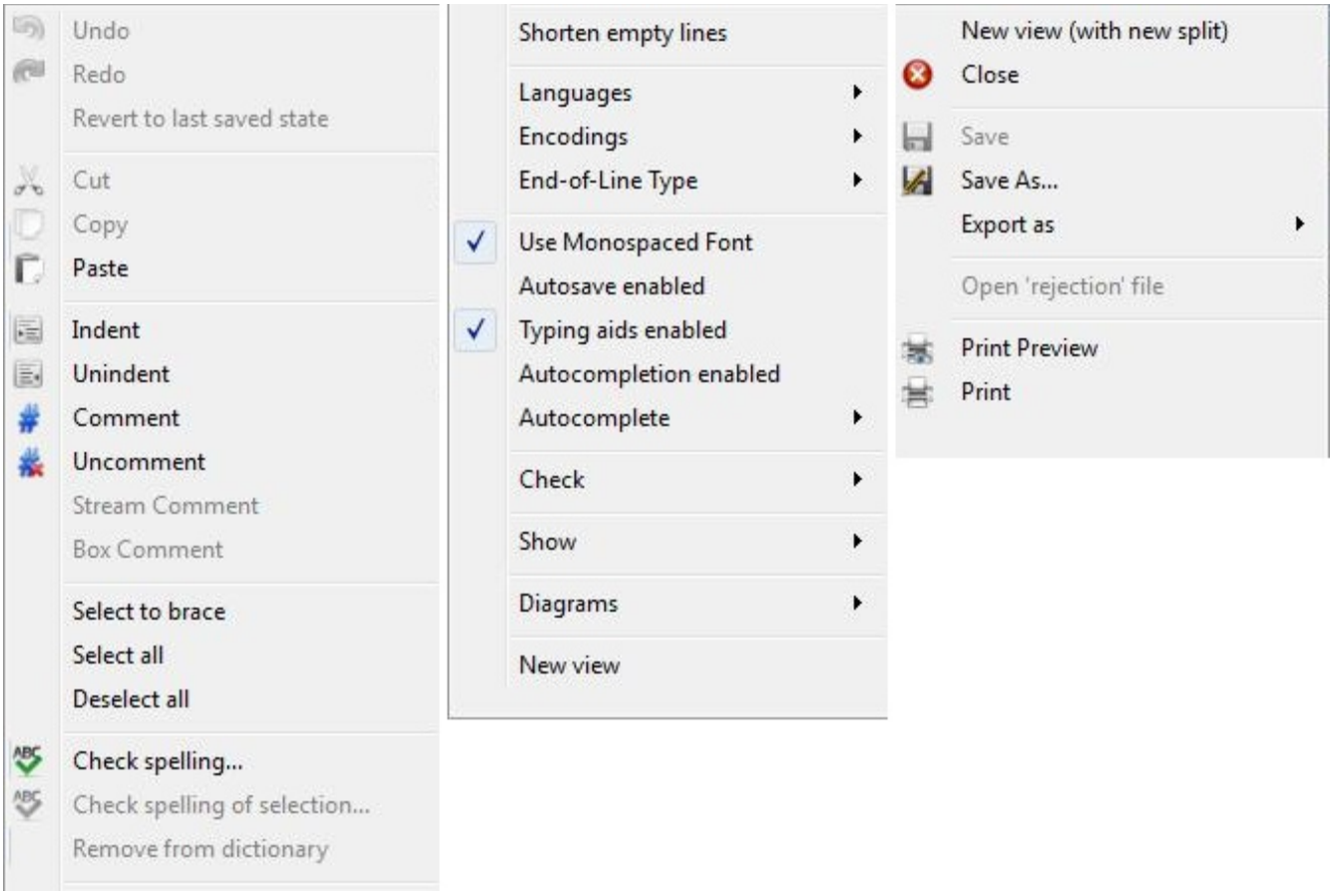
Tab (^) **Copy Path to Clipboard**

Designed to copy into the system clipboard the currently selected item's path, ready to be then "pasted" somewhere. A little handy feature useful when such an argument is then required, and no search button is in sight.

-<>-

Text Form (^) Menu

Normal pop-up context menu 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[80] on the `Settings` > `Preferences... - Editor > Style, Configure editor styles` control form [*see*].

– –

80 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.

Command List

```
Undo          Redo              Revert to Last Saved State          [see menu: Edit >]
Cut           Copy              Paste        Indent                 [see menu: Edit >]
Unindent      Comment           Uncomment    Stream Comment         [see menu: Edit >]
Box Comment   Select to Brace   Select All   Deselect All           [see menu: Edit >]
Check Spelling...                                        [see: Extras > Check Spelling...]
Check Spelling of Selection...   Remove from Dictionary
Shorten Empty Lines                                                 [see menu: Edit >]
Languages         Encodings             End-Of-Line Type       Use Monospaced Font
Autosave Enabled  Typing Aids Enabled   Autocompletion Enabled
Autocomplete                                                        [see menu: Edit >]
Check                                                            [see menu: Project >]
Show
Diagrams[81]                                                     [see menu: Project >]
New View    New View (with New Split)
Close       Save        Save As...  Export As                       [see menu: File >]
Open 'Rejection' File                                          [see context: Tab (^)]
Print Preview           Print                                       [see menu: File >]
```

Remark

Description of context commands identical or equivalent to those that can be found in the main Menu Bar [*see in:* {1North}] 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.

– –

81 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.

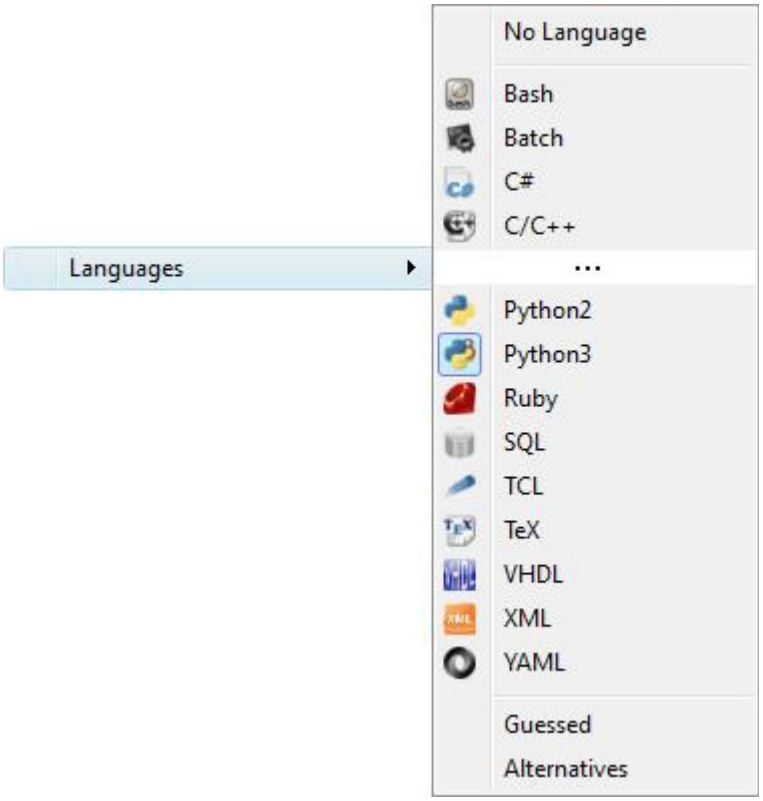
Text Form (^) **Remove from Dictionary**

Designed to do the inverse action of the context command `Spell (^) Add to Dictionary` [*see*] 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 would the automatic detection fail [*cf. section:* How to Select the Language Interpreter, *in:* {0Lead}].

Then the icon of the editing aids language currently enabled is also shown at left on the Information Bar, at the bottom of the Eric Window [*cf. in:* {Map}].

E.g. for Python 3:

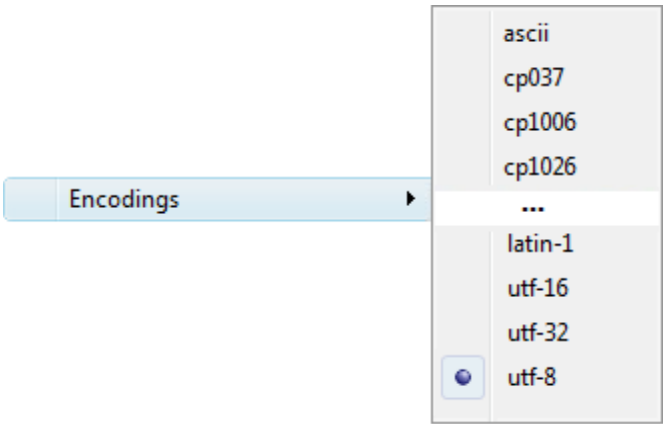
Note that this language selection doesn't concern the source execution, but exclusively its editing, and for such aspects as:

- Syntax highlighting
- Autocomplete and Calltip
- Text Typing Aids, and display
- File type setting, and related

– –

Text Form (^) **Encodings**

Designed to show such a list:



where to possibly change which the character encoding code to enable for the current text edit session [*see also:* How to Select the Unicode Encoding, *in:* {0Lead}].

Currently enabled code is shown at left on the Information Bar, at the bottom of the Eric Window, besides the icon of the editing aids language [*cf. in:* {Map}].

E.g.:

– –

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[82] [*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.:

– –

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`].

– –

82 Current text edit session only. As a rule all context commands are tied to the current context scope only.

Text Form (^) **Autosave Enabled**

Designed to possibly enable / disable the timed autosave action occurring at the "`Autosave interval`" time set on the "`Configure file handling settings`", "`Save`" area [*see:* `Settings > Preferences... - Editor > Filehandling`].

– –

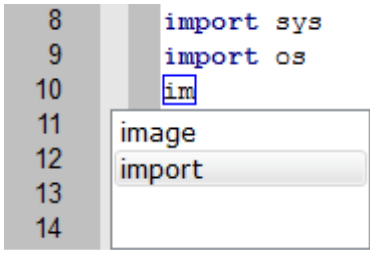
Text Form (^) **Typing Aids Enabled**

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 (^) **Autocompletion Enabled**

Designed to possibly enable / disable such autocompletion function as hereafter exemplified:

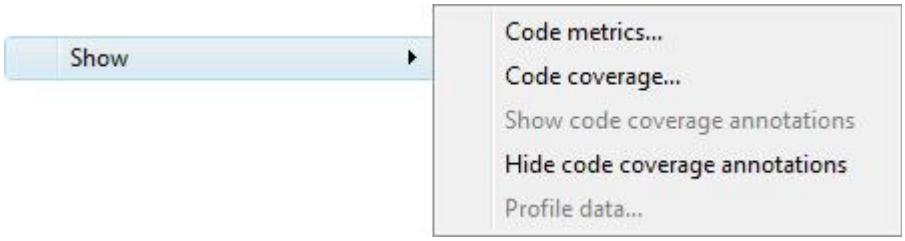


during current text edit session [*see also command:* `Edit > Autocomplete`].

– –

Text Form (^) **Show**

Designed to show this box of sub-menu commands, aimed at controlling the display of specific information, as hereafter detailed.



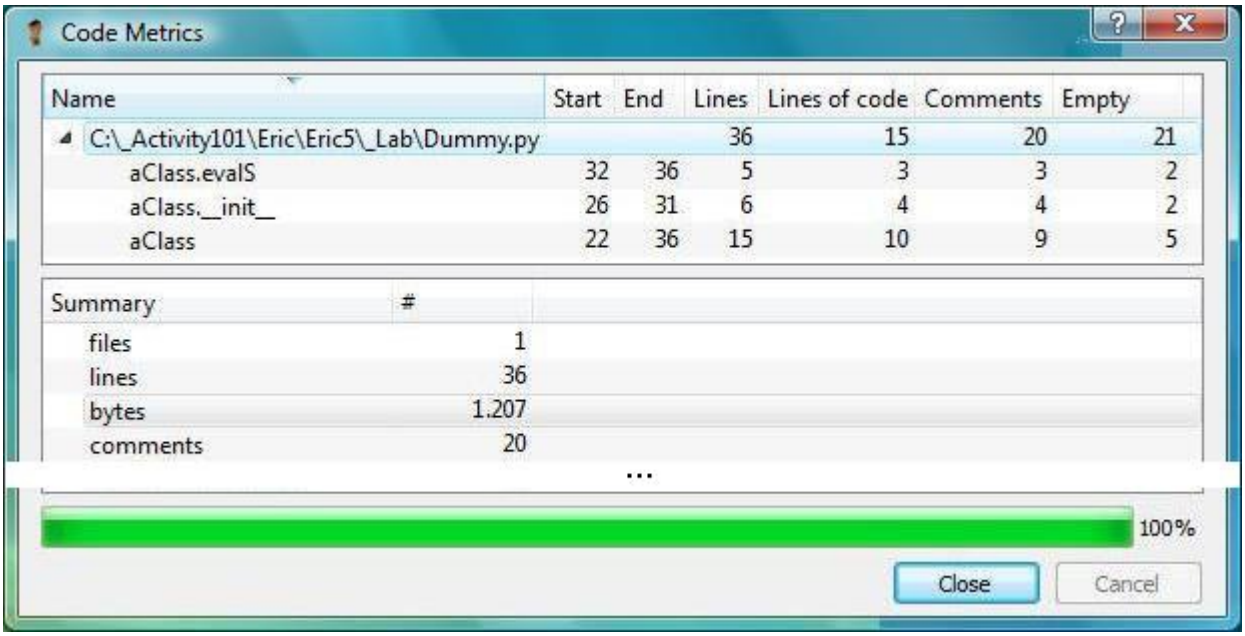
Command List

```
Code Metrics...                          Code Coverage...
Show / Hide Code Coverage Annotations    Profile Data...
```

– –

Text Form (^) Show > **Code Metrics...**

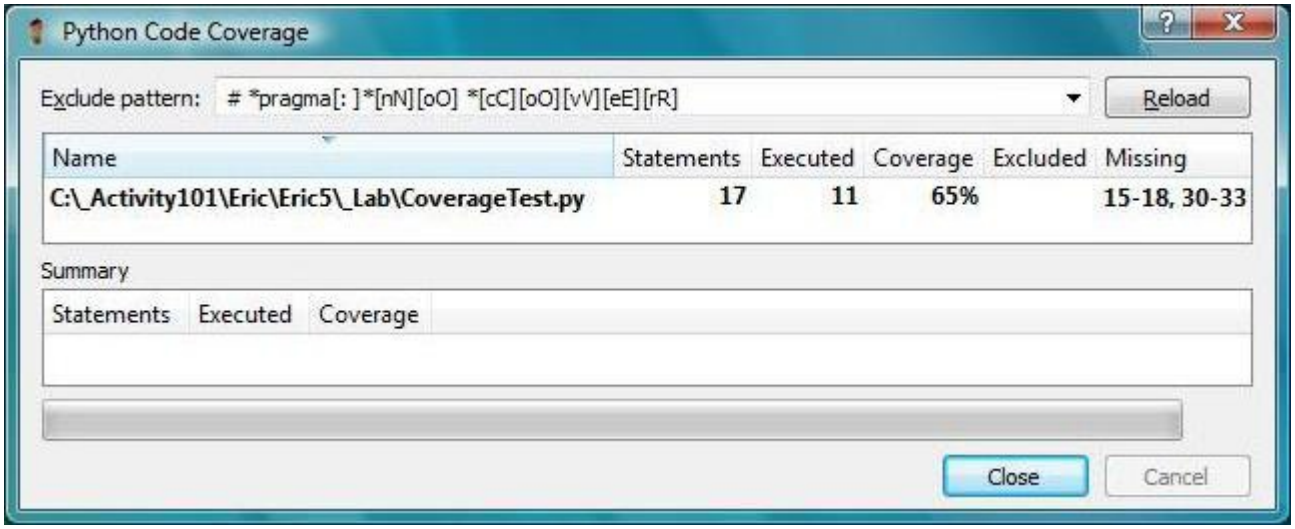
Designed to show such a "`Code Metrics`" box of data, about the current source text.



– –

Text Form (^) Show > **Code Coverage...**

Designed to show such a "`Python Code Coverage`" box, so to display the data collected with a Code Coverage run [*see:* `Start > Coverage Run` *commands*].



Where, in particular:

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

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

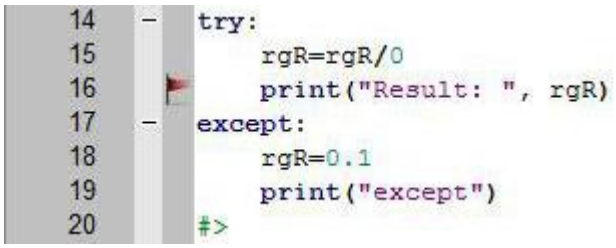
`Summary`
: Area relevant in case of coverage analysis of a Project, not in case of a single module

– –

Text Form (^) Show > Show Code Coverage Annotations

Text Form (^) Show > Hide Code Coverage Annotations

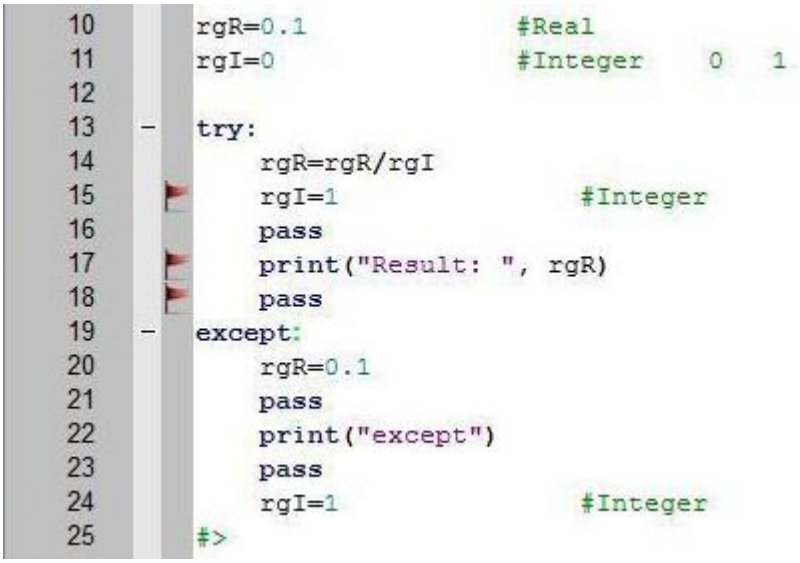
Designed to show / hide the Coverage marks located on the rightmost column of the vertical ruler, according to the currently recorded result of a `Start > Coverage Run` command execution [*see*].



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:



```
10  rgR=0.1             #Real
11  rgI=0               #Integer    0   1
12
13  try:
14      rgR=rgR/rgI
15      rgI=1               #Integer
16      pass
17      print("Result: ", rgR)
18      pass
19  except:
20      rgR=0.1
21      pass
22      print("except")
23      pass
24      rgI=1               #Integer
25  #>
```

a `pass` statement (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 [*see*]. As if a `pass` statement sometimes, but not always, could be treated as it were a comment, or an empty line [*Well, we do not agree*].

– –

Text Form (^) Show > **Profile Data...**

Designed to show Profiling data possibly generated with the `Start > Profile` command [*see*].

– –

Text Form (^) **New View**

Text Form (^) **New View (with New Split)**

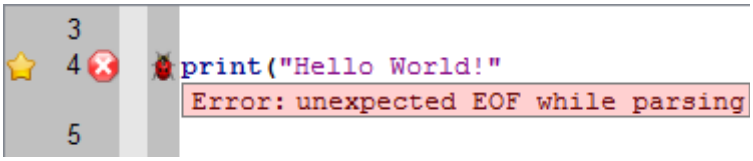
Designed to generate a clone of the current source Text Form, essentially with the purpose of offering two distinct views independently scrollable of the same possibly long source file.

The "`New Split`" version of this command will host the same clone into a new split form, as with command `View > Split View` [*see*].

-<>-

Vertical Ruler, Context Menus

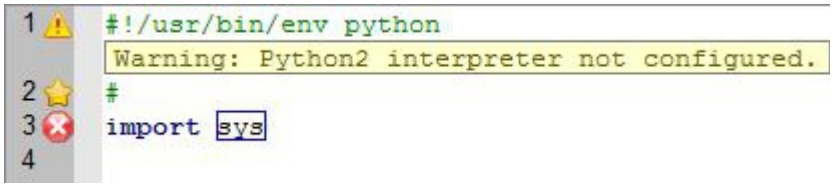
Three distinct pop-up context menus associated to the Vertical Ruler columns,



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`], Global 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`"[83] control box, on the `Settings > Preferences... – Editor > Style, Configure editor styles` control form [*see*].

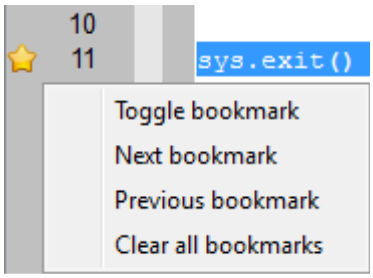


But in this case too, as it is usual in this Report, we'll make reference to the initial default condition only, leaving other possible alternative configurations to the initiative, and preference, of the adventurous user.

– –

83 With "*margin*" that refers to what here we prefer to call "*column*".

Bookmark (^) Menu



Command List

`Toggle Bookmark` `Next Bookmark` `Previous Bookmark` [*see menu:* `Bookmarks >`]
`Clear All Bookmarks`

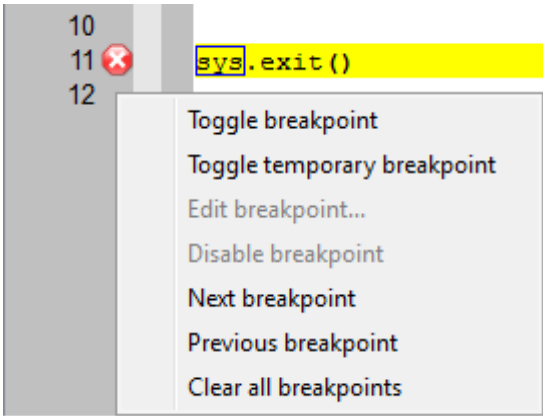
– –

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

`Toggle Breakpoint` [*see menu:* `Debug >`]
`Toggle Temporary Breakpoint`
`Edit Breakpoint...` [*see menu:* `Debug >`]
`Disable Breakpoint`
`Next Breakpoint` `Previous Breakpoint` [*see menu:* `Debug >`]
`Clear All Breakpoints`

– –

Breakpoint (^) **Toggle Temporary Breakpoint**

To set / clear a breakpoint on the corresponding source line. Almost identical to the `Debug > Toggle Breakpoint` command, but for the automatic setting of the "`Temporary Breakpoint`" control check-box.

We recall that Temporary Breakpoints are meant to be executed once, then disabled [*see:* Debug > Edit Breakpoint...].

– –

Breakpoint (^) **Disable Breakpoint**

To disable the currently pointed breakpoint, without clearing it.

– –

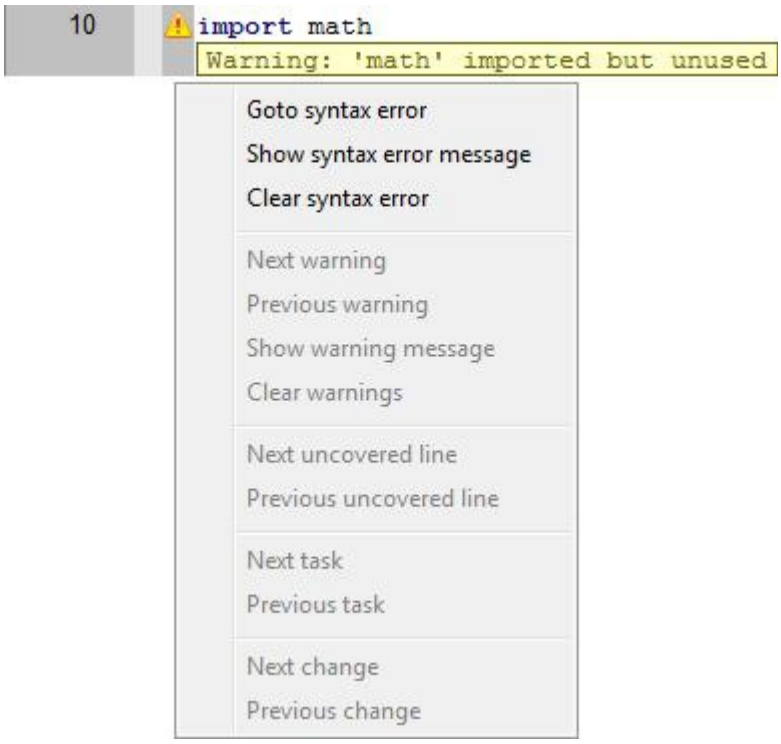
Breakpoint (^) **Clear All Breakpoints**

To clear all breakpoints defined on the active text edit form [*cf. command:* `Debug > Clear Breakpoints`].

– –

Notice (^) Menu

Context menu on the rightmost column of the vertical ruler, where such "Notice" marks as for syntax errors and warnings will be shown.



Command List

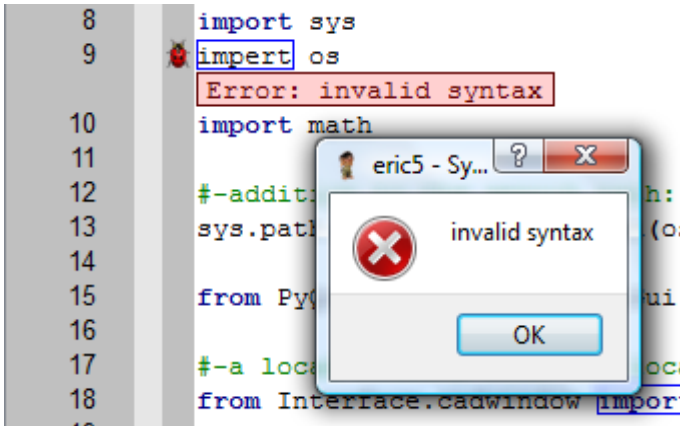
`Goto Syntax Error`		[*see menu:* `Bookmarks >`]
`Show Syntax Error Message`	`Clear Syntax Error`	
`Next Warning [Message]`	`Previous Warning [Message]`	[*see menu*[84]*:* `Bookmarks >`]
`Show Warning Message`	`Clear Warnings`	
`Next Uncovered Line`	`Previous Uncovered Line`	[*see menu:* `Bookmarks >`]
`Next Task`	`Previous Task`	[*see menu:* `Bookmarks >`]
`Next Change`	`Previous Change`	[*see menu:* `Bookmarks >`]

– –

84 <~> Same commands, just with a slightly different denomination [*a difference we 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, 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 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 a next `File > Save` command[85] will anyhow refresh the Python syntax checking, and the consequent error display.

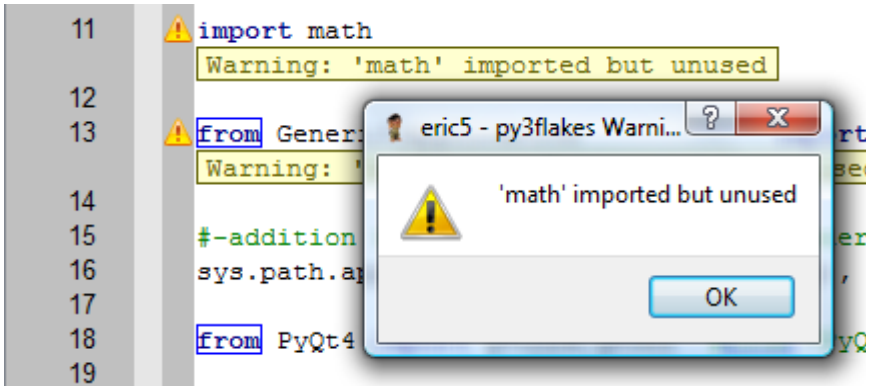
– –

85 Hint: to 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.

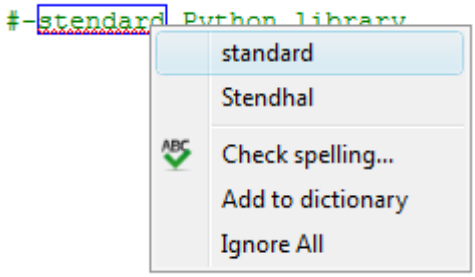


Note that the detection of a syntax error will take precedence over all warning conditions.

-<>-

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. 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...`[86], 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>***

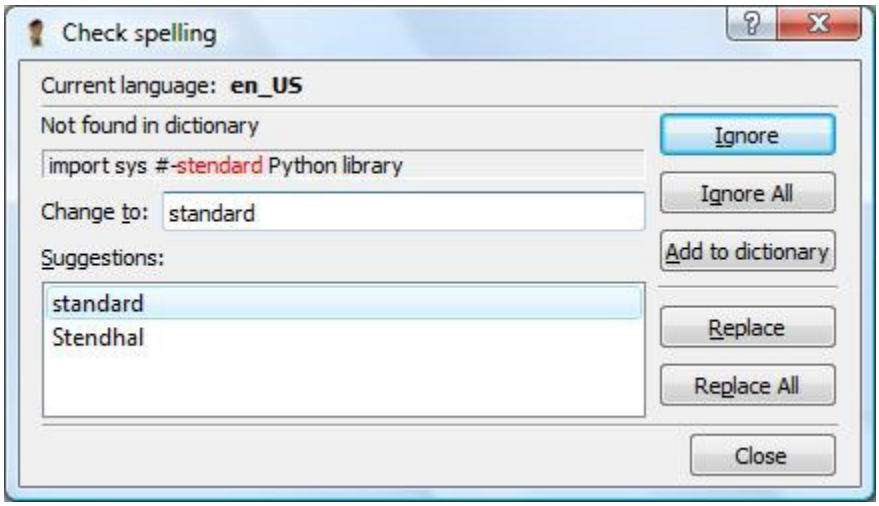
List of spelling aids available to be possibly substituted to the pointed word.

– –

86 Functionally very similar but for the scope of the check, here focused on the single pointed word.

Spell (^) **Check Spelling...**

Designed to show this "`Check spelling`" control box:



where to manage the currently pointed spelling issue, as already described at the functionally similar menu command: `Extras > Check Spelling...` [*see*].

– –

Spell (^) **Add to Dictionary**

Command functionally identical to the corresponding button on the "`Check spelling`" control box [*see*]. 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*], designed to ignore all occurrences of the pointed spelling issue on the current text, and during the current edit session only[87]. It refers to the currently pointed word squiggly underlined.

– = –

87 Therefore in this case there is no need for any dictionary management feature, such as list and delete.

{3South} Eric Window – *South Side*

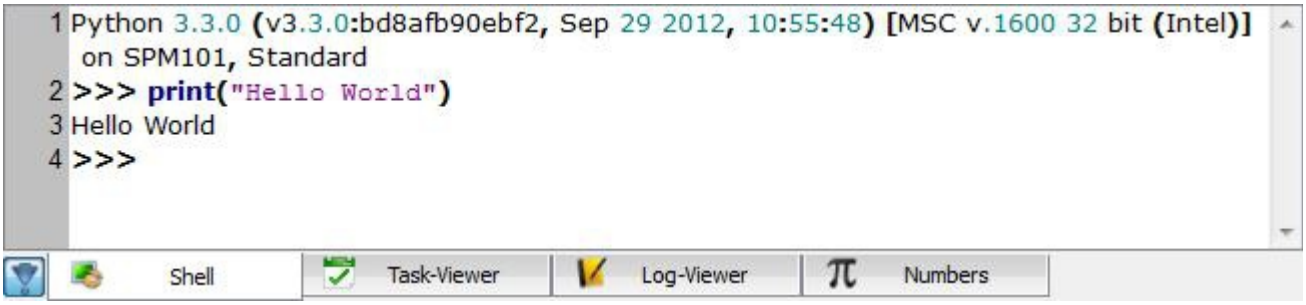
S-PM 130600

Auxiliary Bottom Pane

An optional tabbed form positioned at the bottom of the Central Area, hosting the following set of tabs, ordered from left to right:

```
Shell, Task-Viewer, Log-Viewer, Numbers;
```

described in detail on the next sections.



Remark

The presence of this Bottom Pane[88] is optional in the sense that its presence can be controlled with the main menu command `Window > Bottom Sidebar` [*see*]. And then, when in display, it offers also this toggle-control:

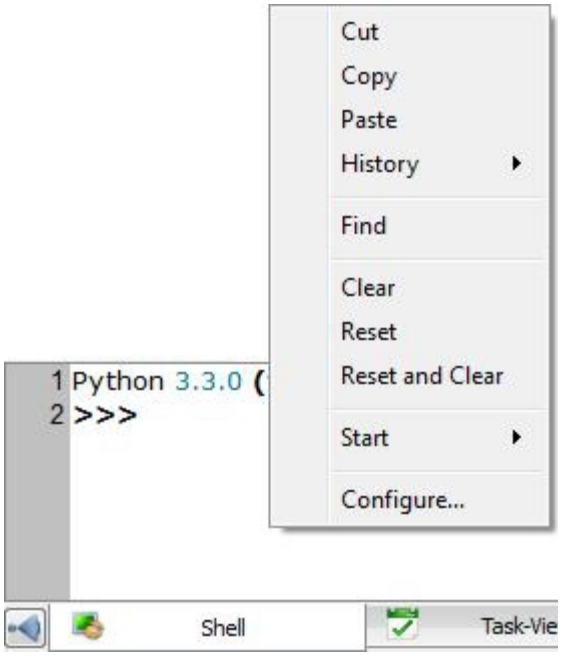
so to possibly activate an automatic collapsing mechanism.

– –

88 "B-Pane" hereafter, for short.

B-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

`Cut` `Copy` `Paste` [*see menu:* `Edit >`]
`History`
`Find` [*see menu:* `Edit > Search`]
`Clear` `Reset` `Reset and Clear` `Start` `Configure...`

– –

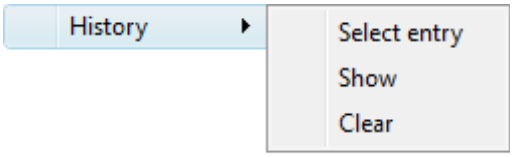
Remark

Description of context commands identical or equivalent to those that can be found in the main Menu Bar [*see in:* {1North}] 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 call a sub-menu aimed at making use of the collection of Python commands last entered into the shell.



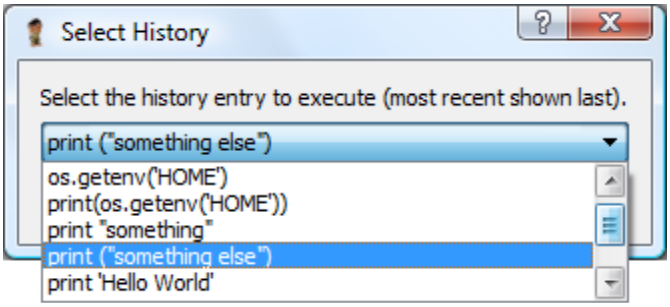
Command List

`Select Entry` `Show` `Clear`

– –

Shell (^) History > **Select Entry**

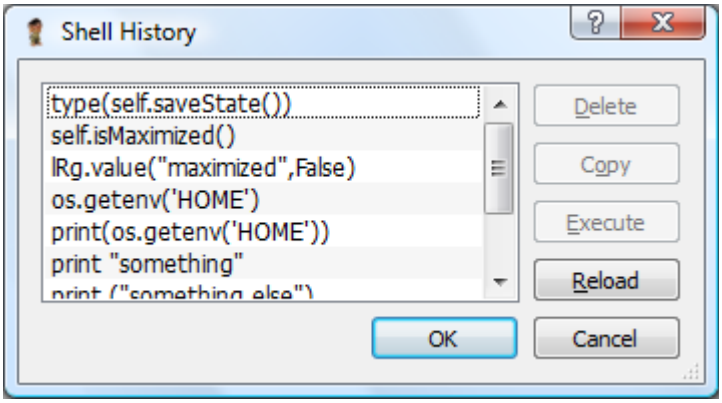
Designed to show this "`Select History`" drop-down list of the last entered Python commands, from which to possibly chose one to be entered again.



– –

Shell (^) History > **Show**

Designed to show this "`Shell History`" control box aimed at managing the list of the last entered Python commands.



– –

Shell (^) History > **Clear**

Designed to erase the list of the last entered Python commands.

– –

Shell (^) **Clear**

Designed to clear the Python command shell contents, brought 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, possibly beyond Python 3 which is an Eric (5) mandatory prerequisite.

Selected language will then remain available at next Eric re-start.

Remark

<!> This means, in particular, that Eric (5) 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*], but conveniently opened on the "`Configure Shell`" control box.

-<>-

B-Pane: Task-Viewer

"`Task-Viewer`" tab form, where to show and manage the so called "*Tasks*", that is the user annotations for actions scheduled to be carried on in future.

There are two different types of such Tasks:

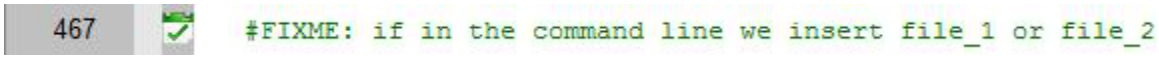
Global Task Developments appointed as to be done in general, i.e.: not specifically tied to a given open Project [*see also commands:* `Bookmarks > Next / Previous Task`].

Project Task Developments specifically tied to the given open Project [*see context:* `Task (^) New Task...`].

– –

Remark

Precise format of the prefix for these special Global Task comment lines, such as:



and:

is as stated on: `Settings > Preferences... – Tasks, Tasks Markers` [*see*].

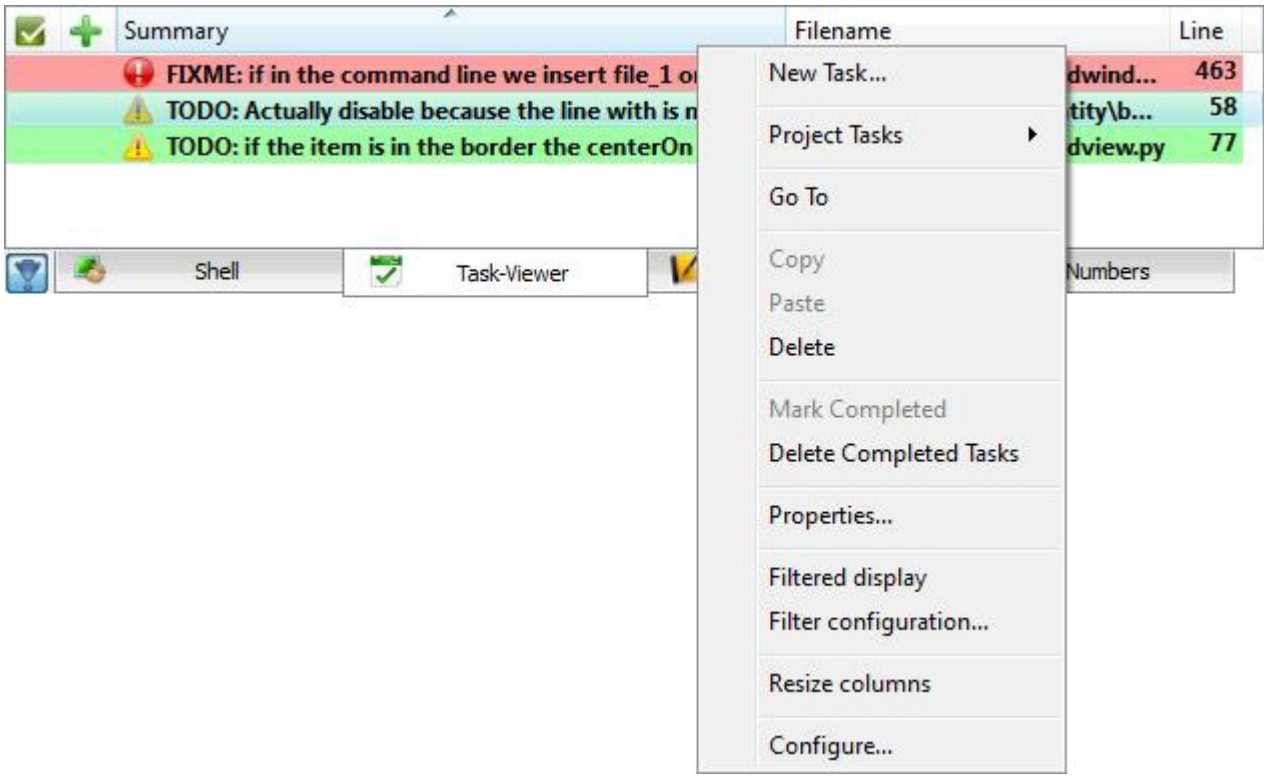
Then a `File > Save` command is required to update the detection of all Tasks possibly last entered on the current software module, and also the consequent display of the related Task Markers on the Notice column [*see also:* Vertical Ruler, Context Menus, *in:* {2Central}].

– –

Task-Viewer Form, and Context Menu

A form for displaying the Tasks Table, possibly listing both Global and Project Tasks. Meaning of the table's columns is assumed as evident.

Just note that the "`Line`" and "`Filename`" columns are significant only for Global Tasks, whereas the "✓" tick-sign (Task Completed) and "+" plus-sign (Task Priority) columns are significant only for the Project's ones [*see:* `Task (^) Properties...`].



Command List

`New Task...`[89] `Project Tasks` `Go To`

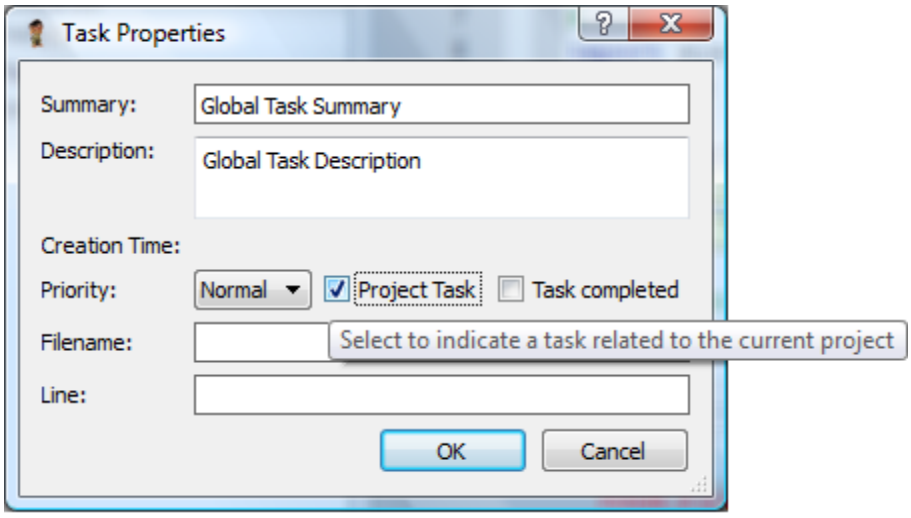
`Copy` `Paste` `Delete` [*see menu:* `Edit >`]
`Mark Completed` `Delete Completed Tasks` `Properties...`
`Filtered Display` `Filter Configuration...` `Resize Columns` `Configure...`

– –

89 <~> The Task here intended is a “Project Task”, only. Whereas with such other command as `Bookmarks > Next Task` [*see*] the task to be intended is a “Global Task”. Confusion & ambiguity that we suggest should be fixed.

Task (^) New Task...[90]

Designed to show a "`Task Properties`" dialog box



where to define a new *Project Task*, that is a Task different in scope from the `#TODO:` and `#FIXME:` *Global Task* as for ***B-Pane:*** Task-Viewer above section [*see*]. Project Tasks are intended as annotations for developments specifically tied to a given Project.

Note that this "`Task Properties`" box is the same as with `Task (^) Properties...`, [*see*], with the `Filename` and `Line` fields here disabled as not significant in this Project's case.

– –

Task (^) Project Tasks

Designed to show a box of sub-menu commands aimed at managing the generation of this Task Table in case of an opened Project.

– –

90 <~> Here to be intended "Project Task" only, not "Global Task".

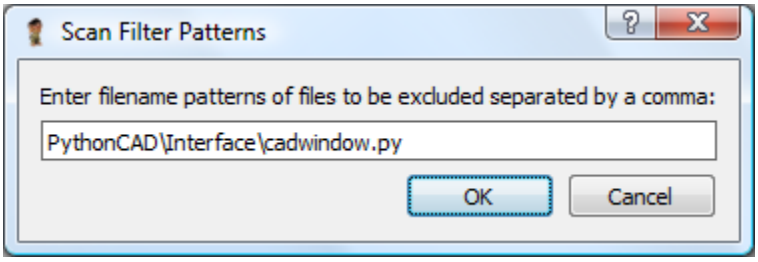
Task (^) Project Tasks > **Regenerate Project Tasks**

Designed to refresh this Task Table according to the current condition, in particular after a new `Project Tasks > Configure Scan Options` [*see next context command*].

– –

Task (^) Project Tasks > **Configure Scan Options**

Designed to show a "`Scan Filter Patterns`" control box



where to enter the filename patterns to be excluded from the Task table, possibly comma separated and with usual "`*`" wildcards; e.g.: "`ThirdParty\*,*\coverage\*`". The changes entered then require a: `Project Tasks > Regenerate Project Tasks` to take effect [*see previous context command*].

Last Filter Patterns entered will be then in display when calling over this command.

– –

Task (^) **Go To**

Designed to synchronize the Python module in display on the source text form with the selected Task line. A double-click on the very Task table line will have the same result.

– –

Task (^) **Mark Completed**

Designed to mark the selected Project Task as "Completed"[91]. Completion mark can be reset using the context command `Properties...` [*see*].

– –

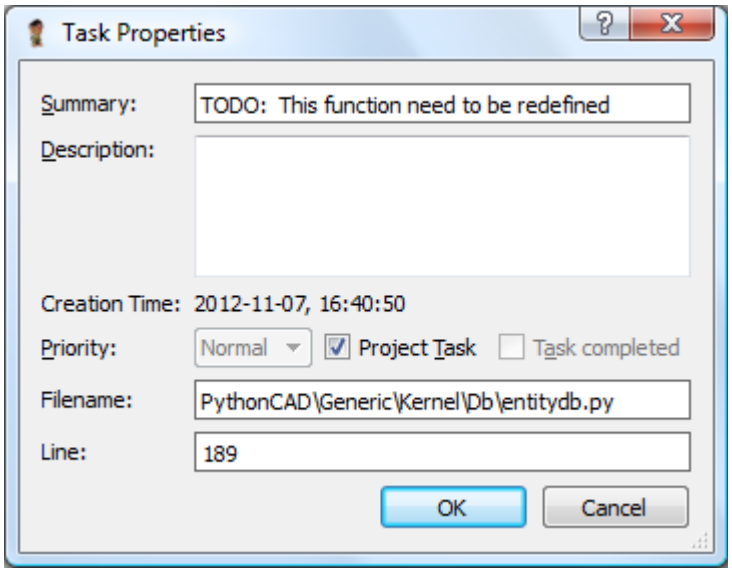
Task (^) **Delete Completed Tasks**

Designed to delete all Project Tasks currently marked as Completed [*cf. command:* `Task (^) Mark Completed`].

– –

Task (^) **Properties...**

Designed to show this "`Task Properties`" dialog box



where to see and manage the current Task's properties, either Global or Project. Read-only, read-write and enabled fields will properly vary in case of Global or Project Tasks.

– –

91 Global Tasks do not need such commands, as can be managed acting directly at source code level.

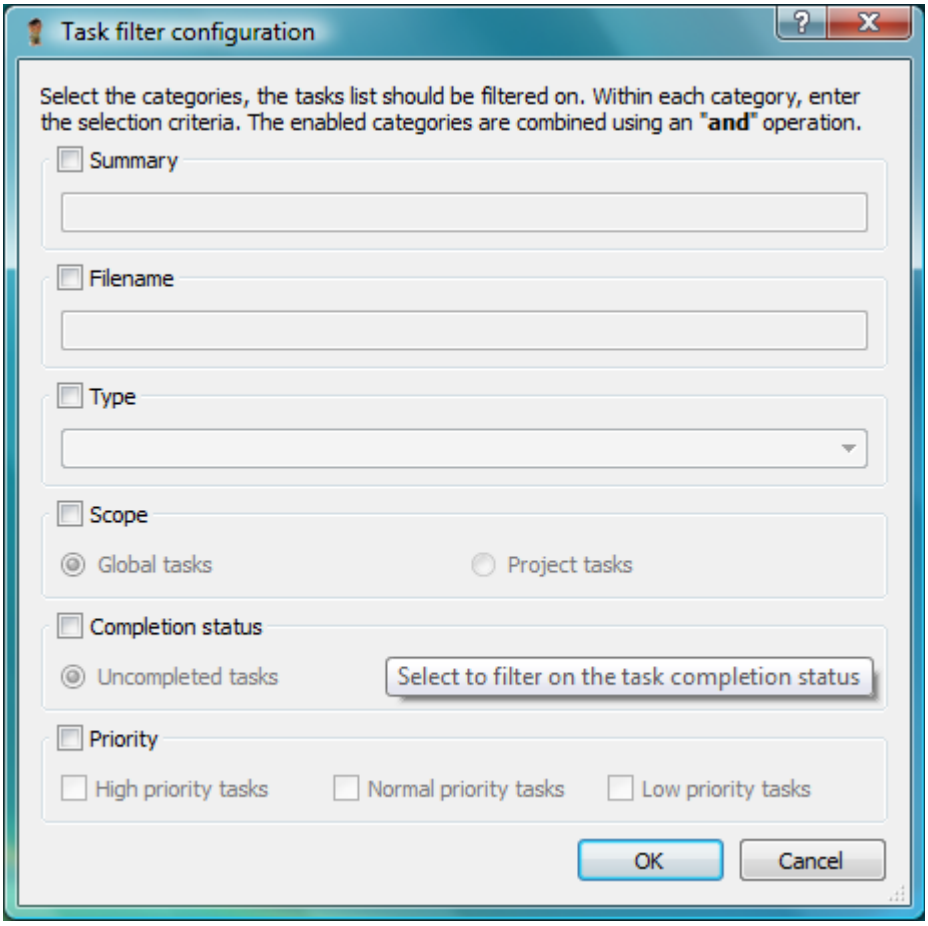
Task (^) **Filtered Display**

Designed to enable / disable the `Task (^) Filter Configuration...` [*see*] currently defined.

– –

Task (^) **Filter Configuration...**

Designed to show this "`Task filter configuration`" control box:



where to set selection criteria for the Tasks to be listed. Enabled / disabled fields can vary according to the very Task condition.

Anyhow this is a control box assumed clear enough not to require any further explanation.

– –

Task (^) **Resize Columns**

Designed to reset the automatic width of the Task-Viewer table's columns, possibly hand-modified.

– –

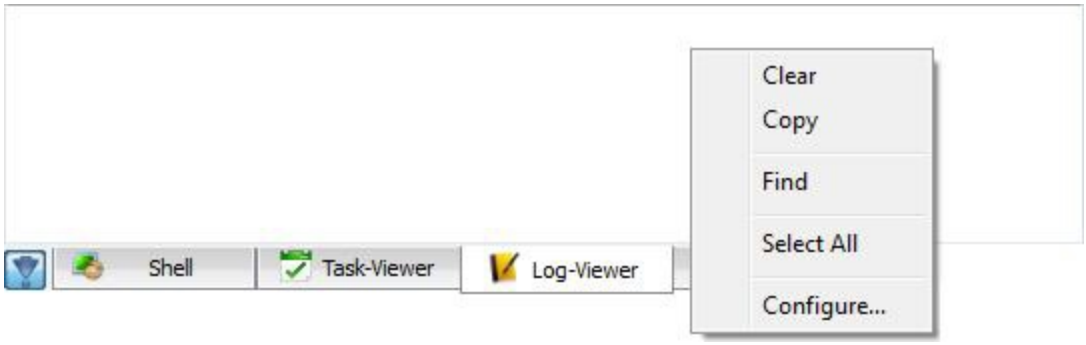
Task (^) **Configure...**

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure Tasks`" control box.

-<>-

B-Pane: Log-Viewer

"`Log-Viewer`" tab form, where Eric is assumed to print and show its own "Standard Output" and "Standard Error" messages[92], 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[93]. 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*], that is not here.

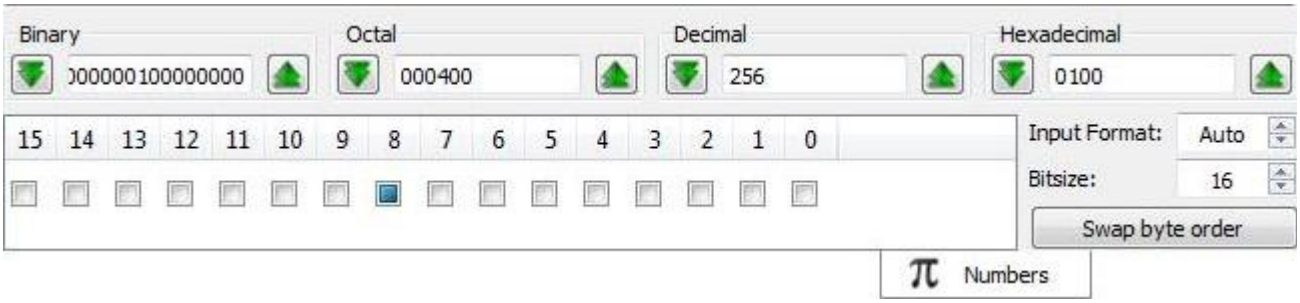
-<>-

92 That is: "`stdout`" and "`stderr`" devices, saying it with usual Unix terms.

93 Such as, we've been told: PyLupdate and iRelease, when dealing with translation strings.

B-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

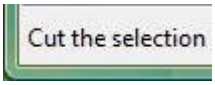
`Swap byte order` Button provided for a "`Bitsize`" value greater than 8 [*see*].

-<>-

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



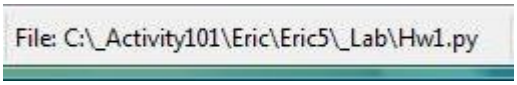
Eric information message, to the operator

Syntax highlighter language [c*f.:* `Text Form (^) Languages`, *in:* {2Central}]

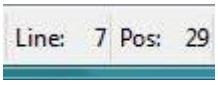
Character encoding code [c*f.:* `Text Form (^) Encodings`, *in:* {2Central}]

End-Of-Line Type [c*f.:* `Text Form (^) End-Of-Line Type`, *in:* {2Central}]

Read-Write ("`rw`") or Read-Only ("`ro`"), as current text file permission



Current text file path



Current line & character insertion point position

VCS Status Monitor LED [*see command:* `Project > Version Control`].
Color coded as in `Shift+F1` Help Hint [*see*].

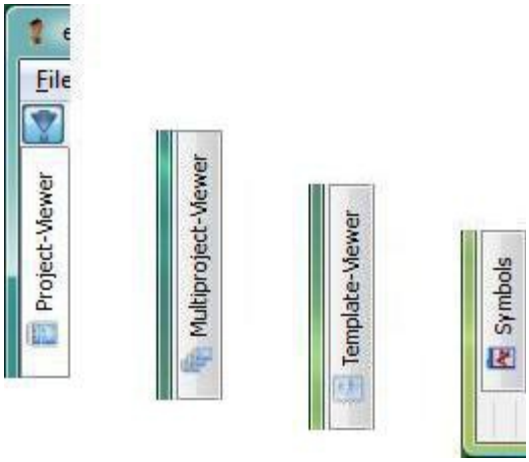
– = –

{4West} Eric Window – *West Side*

Auxiliary Left Pane

An optional pane positioned at left on the Eric window[94], 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, Symbols`



Optional in the sense that its presence can be controlled with the main menu command `Window > Left Sidebar`, selected with `Window > Edit Profile / Debug Profile` commands and 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.

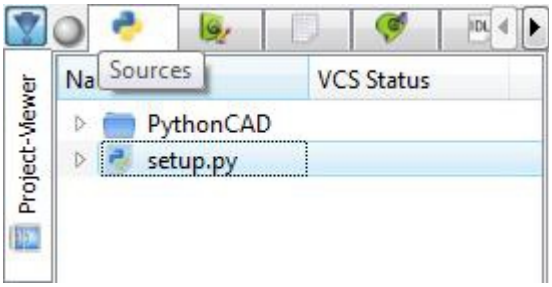
– –

94 "L-Pane" hereafter, for short.

L-Pane: **Project-Viewer**

A tabbed form, hosting the following set of tabs ordered from left to right, described in detail on subsequent sections:

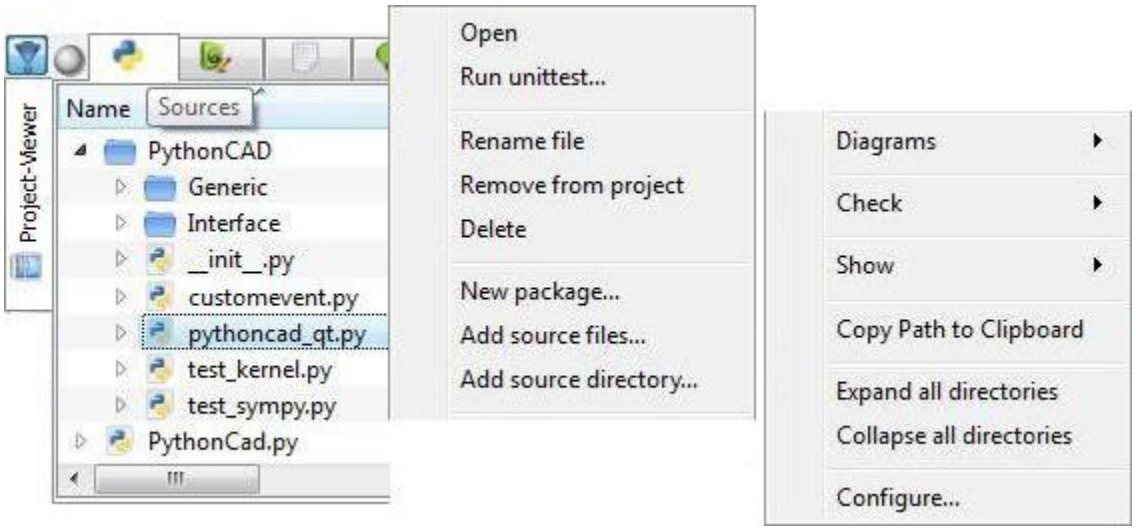
`Sources, Forms, Resources, Translations, Interfaces (IDL), Others`



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

```
Open                                                    [see menu: File >]
Run unittest...                                         [see menu: Unittest >]
Rename File             Remove from Project    Delete
New Package...          Add Source Files...    Add Source Directory...
Diagrams[95]            Check                  Show     [see menu: Project >]
Copy Path to Clipboard                  [see context: Tab (^), in: {2Central}]
Expand / Collapse All Directories       Configure...
```

– –

Sources (^) **Rename File**

Designed to rename the selected file. To be used with due care, as dealing with Projects, where file names count, and synchronization of changes is not automatic.

– –

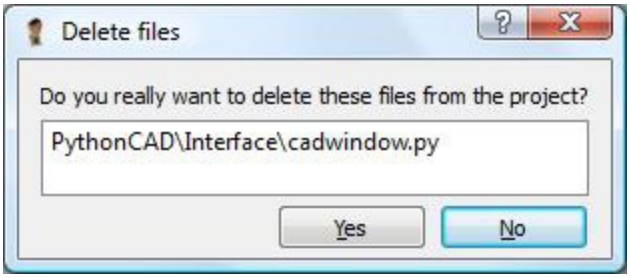
Sources (^) **Remove from Project**

Designed to just exclude the selected file from being in display on the `Sources` list of the `Project-Viewer`, with the very file and its source role that remain unaltered. The original display condition can be anyhow reset by means of command `Sources (^) Add Source Files...` [*see*].

– –

Sources (^) **Delete**

Designed to actually delete the files currently selected on the `Project-Viewer` form.



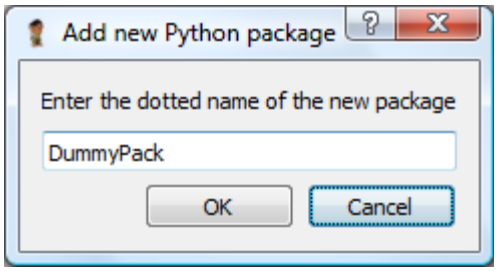
95 For some reason this context sub-menu is richer the the referred main menu correspondent, as it comprises these other sub-commands: `Class / Package / Imports Diagrams...` [*see*]. Anyhow we still consider this as a specialized topic, out of this Report's scope, and about which here we have nothing else to add to what already said.

<!> 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 seems to suggest.

– –

Sources (^) **New Package...**

Designed to show an "`Add new Python package`" control box, so to create a new *Package* as intended by Python.



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 for the `import` Python statement, e.g.: `myLib.subLib`

– –

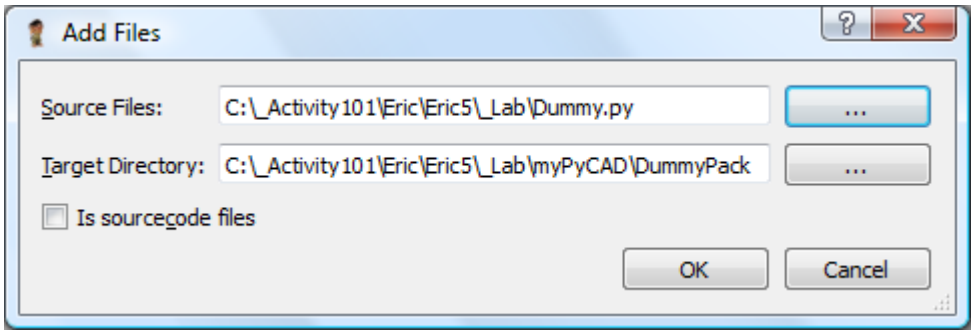
Remark

Then, to possibly fill such a Package with existing source files, next `(^) Add Source Files...` and `(^) Add Source Directory...` context commands 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...`].



Specifications:

`Is sourcecode files`
: Check-box to tell that a file with a not-source type standard extension (e.g. not a: `*.py`) should anyhow be here accepted by Eric as a source file[96].

In case of distinct source and destination directories, the named files are actuallly *copied* into the `Target Directory`, with the original files left unaltered.

Remark

A pair of practical suggestions:

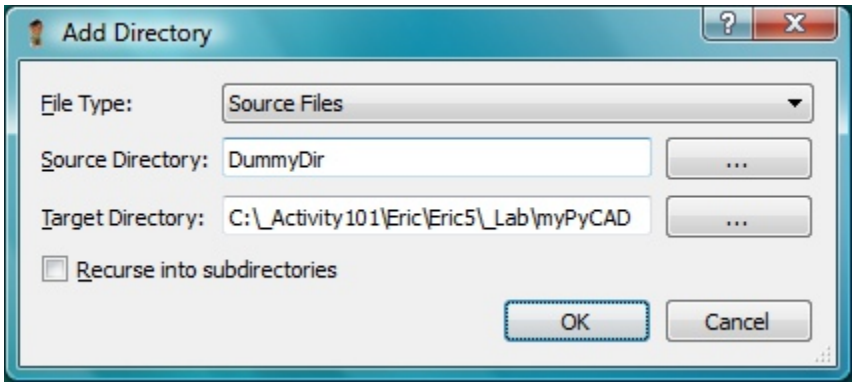
- Not only source files can be here searched and added to a Project, but *ANY* type of file, just selecting the "`*.*`" file extension type when searching the `Source` directory.
- Making use of the search button only one file can be added at a time. Entering full qualified paths directly into the "`Source files`" field, more than one file at a time can be added.

– –

Sources (^) **Add Source Directory...**

Designed to show this "`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...`].

96 <~> At least so we've been told. But, frankly, here we have to confess that we haven't seen any difference whatsoever...



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`.

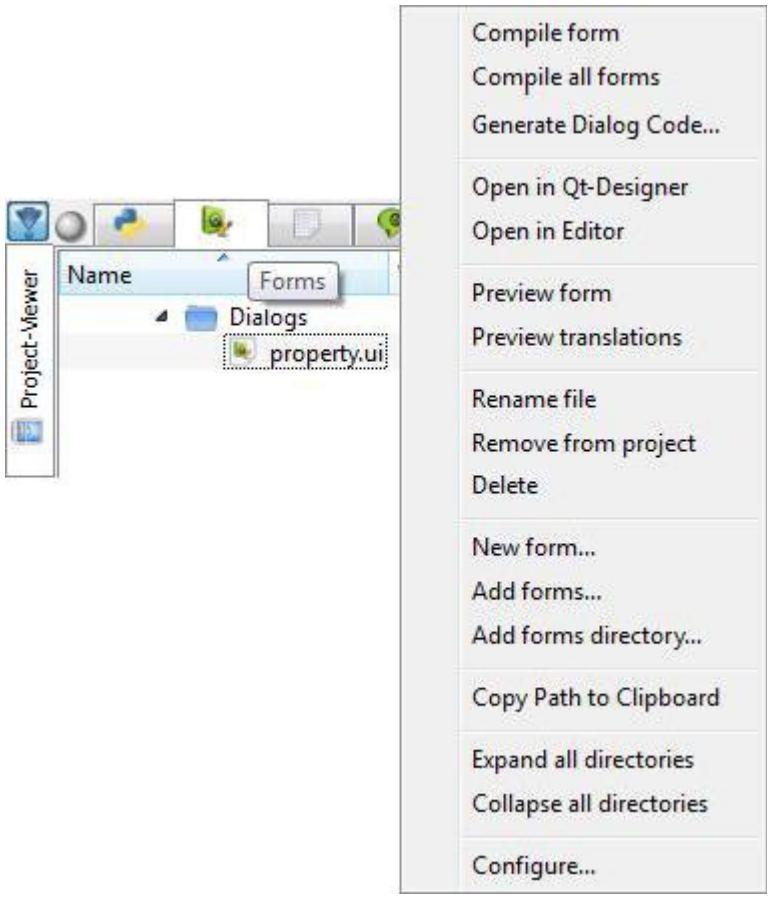
– –

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`
`Rename File` `Remove from Project` `Delete` [*see context:* `Sources (^)`]
`New / Add Forms` `Add Forms Directory...` [*see context:* `Sources (^)`]
`Copy Path to Clipboard` [*see context:* `Tab (^)`, *in:* {2Central}]
`Expand / Collapse All Directories` `Configure...` [*see context:* `Sources (^)`]

– –

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 > Tools (Select Tool Group = Builtin Tools)> Qt-Designer...` [*see*].

Actually a central designing tool equipped with a contour of other tools, as shown in the context-menu [*see*], and whose behavior can be variously enriched and customized with other optional Eric plug-ins [*see:* Plugins Command Menu, *in:* {1North}].

– –

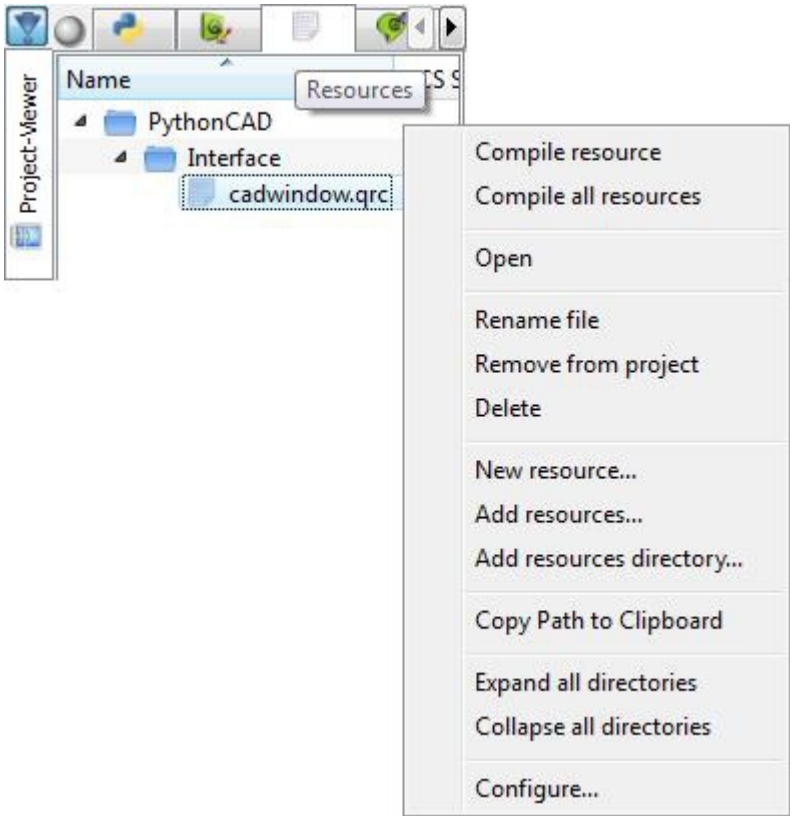
Certainly a relevant and engaging subject, that would deserve a dedicated Eric booklet of its own [*whereas for a general reference see:* Reference Book List, *in:* Appendix], but here just hinted at, as assumed off the scope of this Report [*see:* Scope of this Report, *in:* {0Lead}].

-<>-

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:* {1North}].



Command List

`Compile Resource  Compile All Resources  Open`
`Rename File  Remove from Project  Delete` [*see context:* `Sources (^)`]
`New / Add Resources  Add Resources Directory...` [*see context:* `Sources (^)`]
`Copy Path to Clipboard` [*see context:* `Tab (^)`, *in:* {2Central}]
`Expand / Collapse All Directories  Configure...` [*see context:* `Sources (^)`]

– –

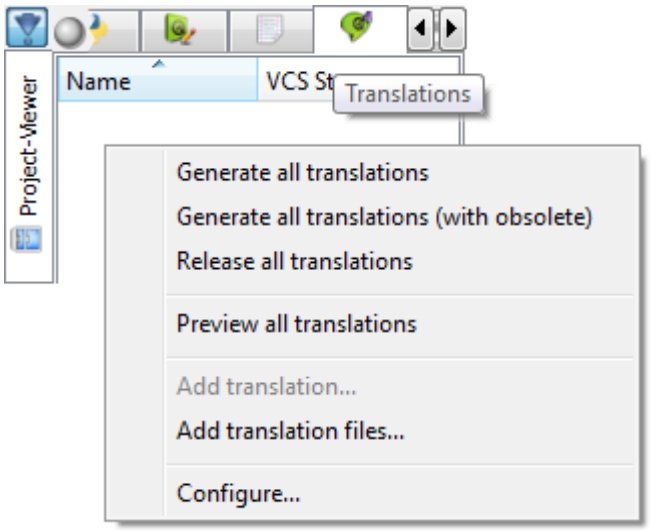
A self-standing subject of specific relevance, here just hinted at, as assumed off scope for this Report [*see:* Scope of this Report, *in:* {0Lead}].

-<>-

Project-Viewer: **Translations**

This Eric feature has mainly to do with Qt Linguist[97], that is a tool for adding translations to Qt applications, also available as an Eric builtin tool [*see:* `Extras > Tools (Select Tool Group = 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:* {1North}].



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

Command List

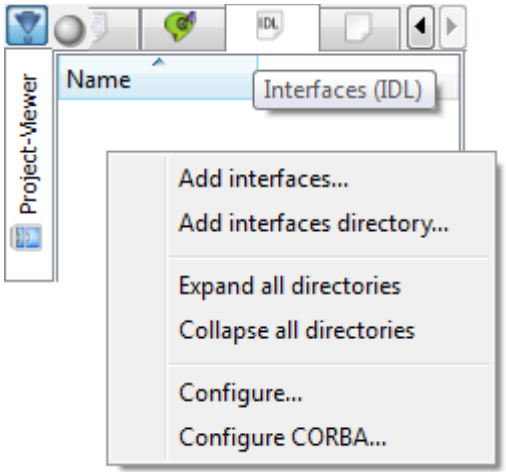
`Generate All Translations` `Generate All Translations (with Obsolete)`
`Release All Translations` `Preview All Translations`
`Add Translation...` `Add Translations Files...` [*see context:* `Sources (^)`]
`Configure...` [*see context:* `Sources (^)`]

-<>-

97 Plus, 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 language.



The 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:* {0Lead}].

Command List

`Add Interfaces...` `Add Interfaces Directory...` [*see context:* `Sources (^)`]
`Expand / Collapse All Directories` `Configure...` [*see context:* `Sources (^)`]
`Configure CORBA...`

– –

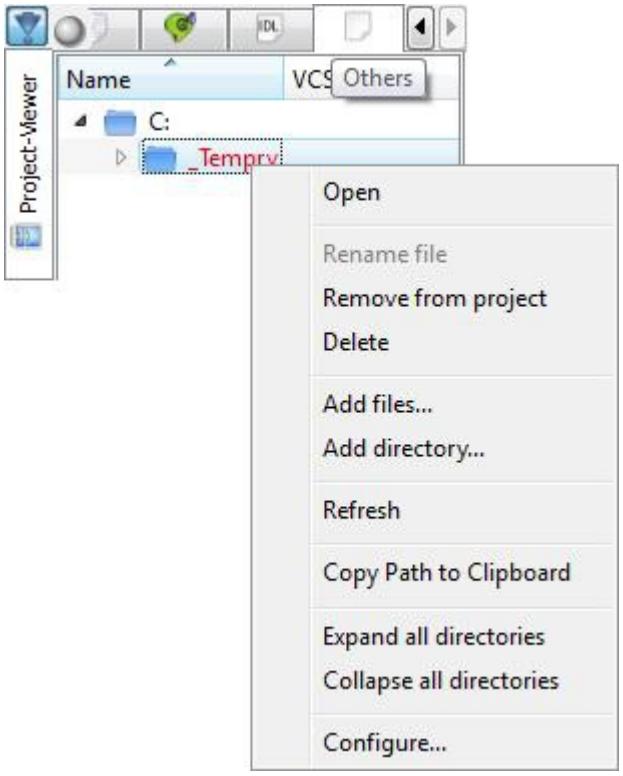
Interfaces (IDL) (^) Configure CORBA...

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

`Open` [*see menu:* `File >`]

`Rename File` `Remove from Project` `Delete` [*see context:* `Sources (^)`]

`Add Files...` `Add Directory...` [*see context:* `Sources (^)`]

`Refresh`

`Copy Path to Clipboard` [*see context:* `Tab (^)`]

`Expand / Collapse All Directories` `Configure...` [*see context:* `Sources (^)`]

– –

Others (^) **Refresh**

Designed to update the contents of this `Project-Viewer: Others` form, as possibly dynamically modified by external actions.

-<>-

L-Pane: **Multiproject-Viewer**

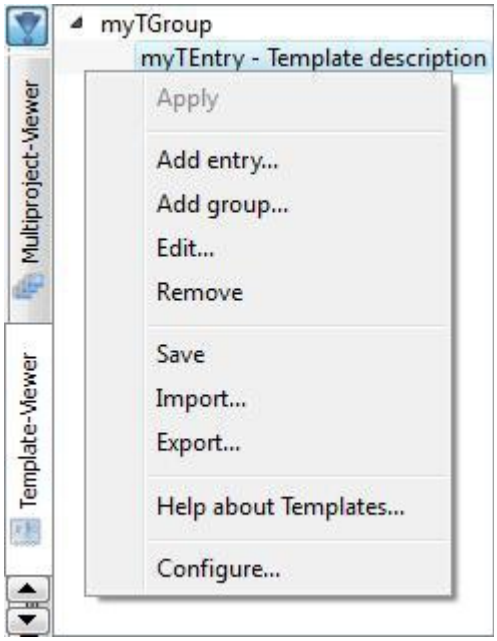
Just an extension of the similar `Project-Viewer` form [*see*], whose reference is to be considered valid here too.

Same way the `Multiproject` main menu item has been considered, and treated, 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 modules [*cf.:* `Extras > Macros`, *similar feature, less powerful*].



Command List

```
Apply      Add Entry...   Add Group...  Edit...                  Remove
Save       Import...      Export...     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 very template will get the same effect.

In case of Template variables [*see command:* `Template (^) Add Entry..., Help` *button*], they will be asked at this moment.

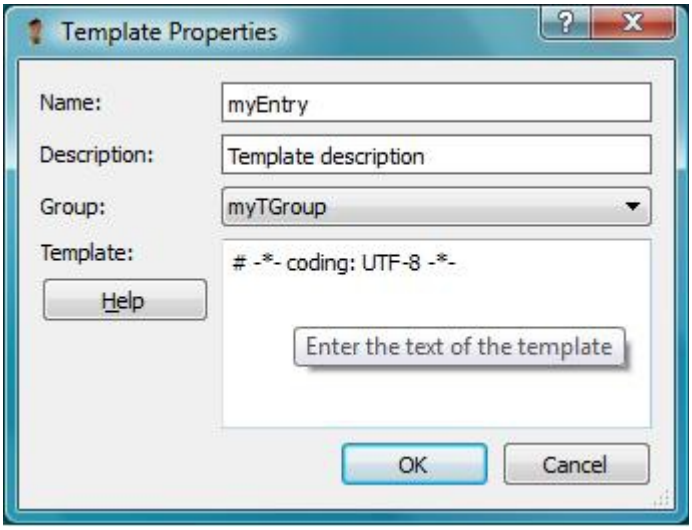
Remark

For an even more direct insertion method see the Remark on the next `Template (^) Add Group...` command.

– –

Template (^) **Add Entry...**

Designed to show this "`Template Properties`" control box:

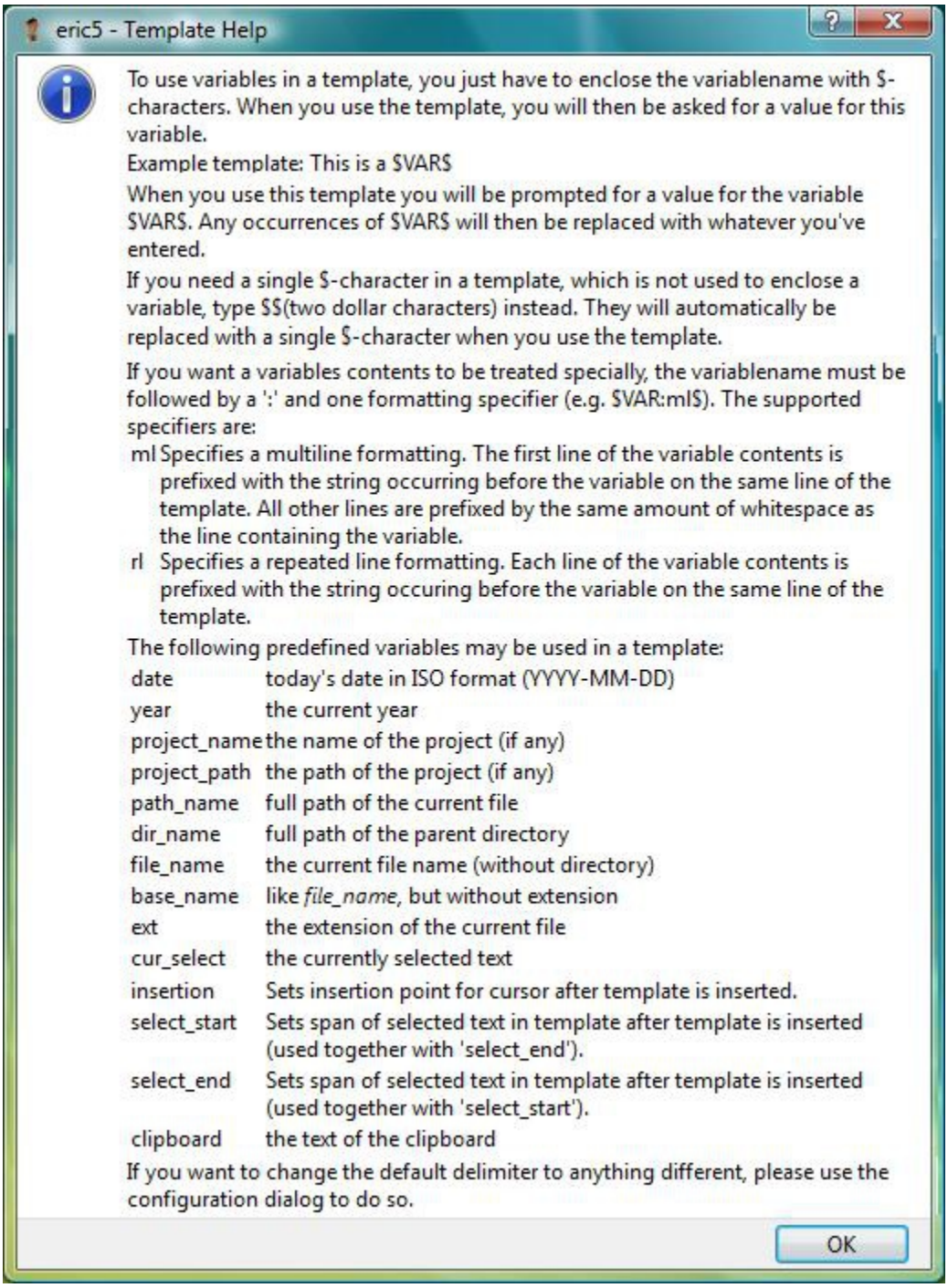


where to create a new Template to be added to an existing Group [*see:* `Template (^) Add Group...`]. Desired source text can be added right away into the `Template` field.

– –

Remark

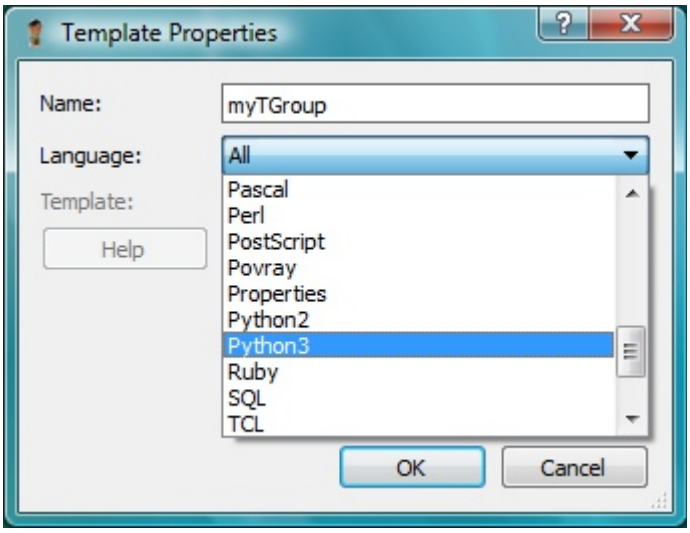
The `Help` button is to display the following essential Eric "`Template Help`" instruction box, useful if you want to know how to create Templates enriched with variables to be resolved at insertion time.


eric5 - 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.
OK

– –

Template (^) **Add Group...**

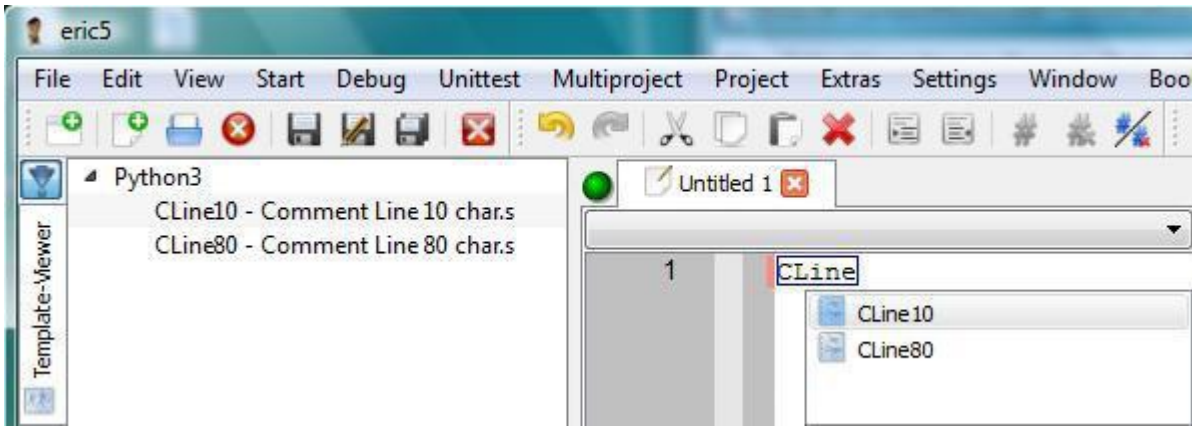
Designed to show this "`Template Properties`" control box:



where to create a Template Group, as a required container for actual Templates. A control box similar to that related to the `Template (^) Add Entry...` command [*see*], but with the `Template` and `Help` controls disabled.

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 example hereafter:



<!> Templates stored into a Group named exactly as the relative 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.

– –

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.

– –

Template (^) **Import...**

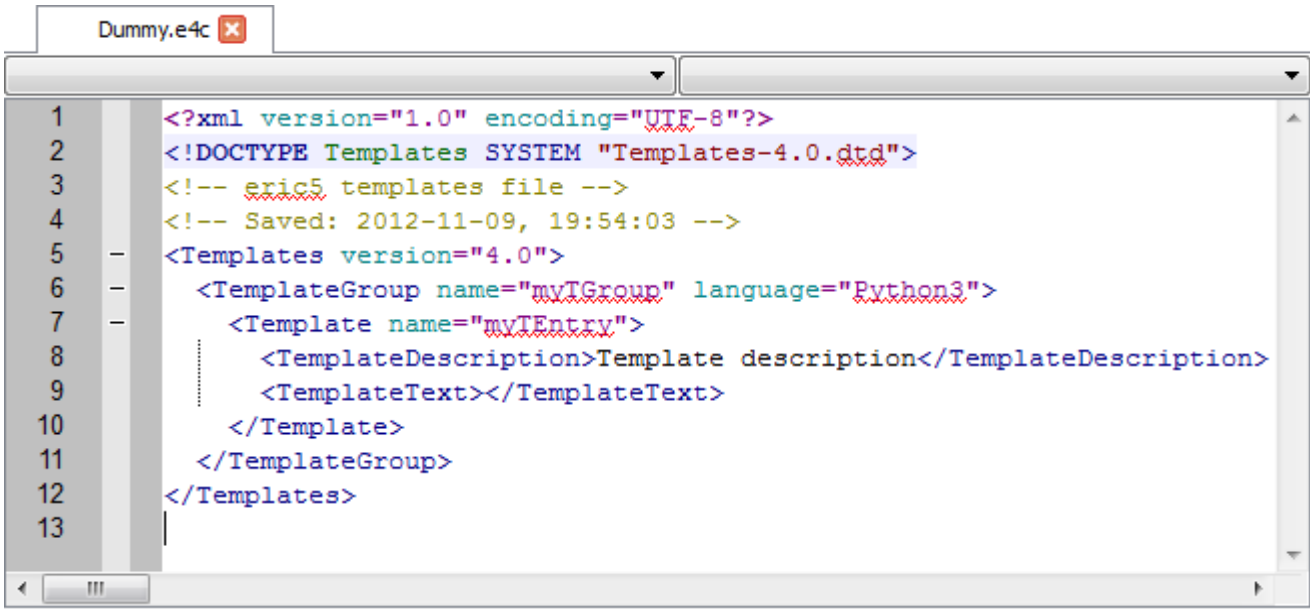
Designed to import an Eric Template collection "`*.e4c`" file, formerly exported with command `Template (^) Export...` [*see*]. The contents of the given file will be then copied and merged into the current `<User>\_eric\eric5templates.e4c` file. Former contents unaltered.

– –

Template (^) **Export...**

Designed export the current Template collection into a XML "`*.e4c`" file, so to be then possibly imported into another Eric environment [*see:* `Template (^) Import...`].

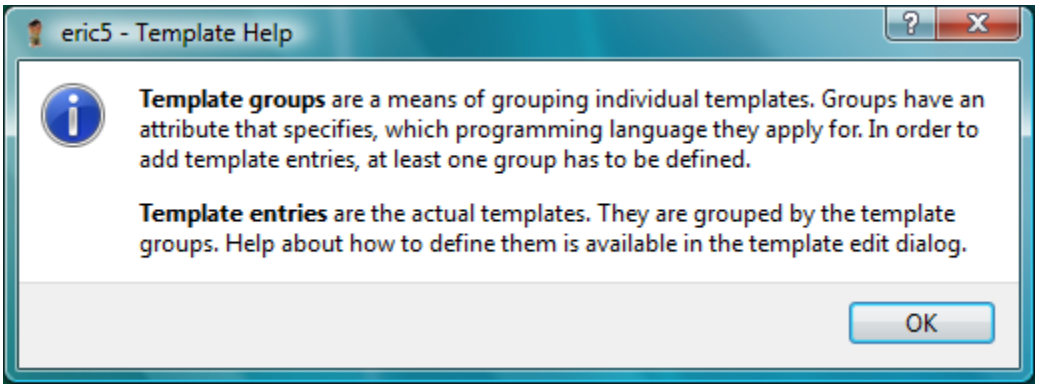
Note that such exported file can be conveniently opened and shown also into the Eric's text edit form.



– –

Template (^) Help about Templates...

Designed just to display this "`Template Help`" dialog box:



– –

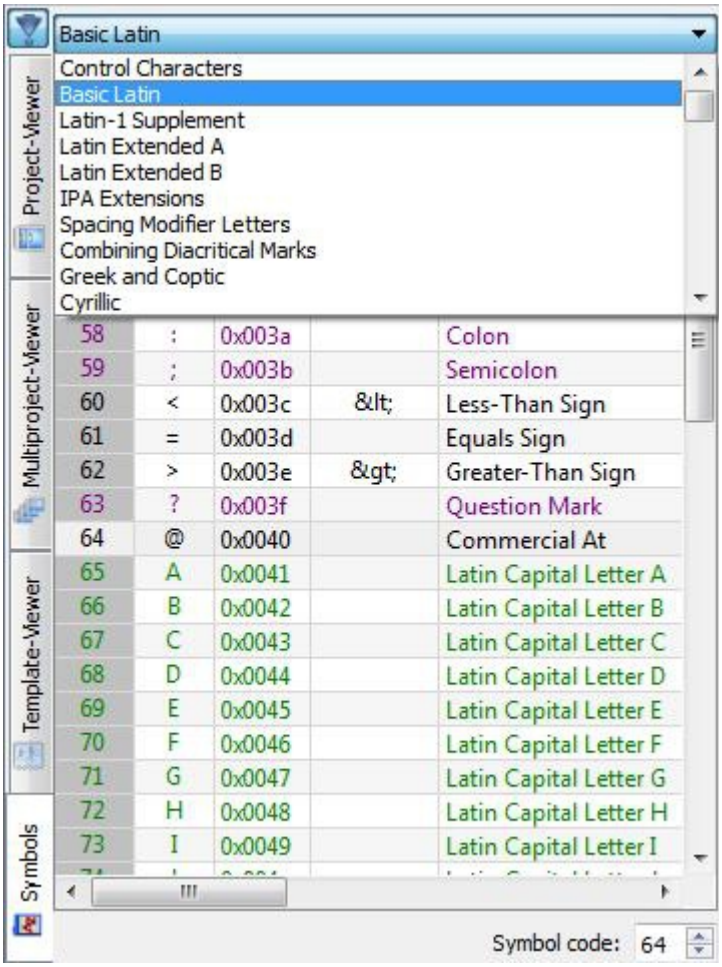
Template (^) Configure...

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure Templates`" control box.

-<>-

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”[98], 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.

– = –

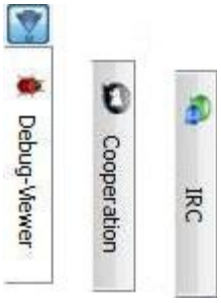
98 Ethiopia is where ancient Gods of the Greek mythology used to go in holiday, and also where we've been lucky enough to live, for some time.

{5East} Eric Window – *East Side*

Auxiliary Right Pane

An optional pane positioned at right on the Eric window[99], 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`



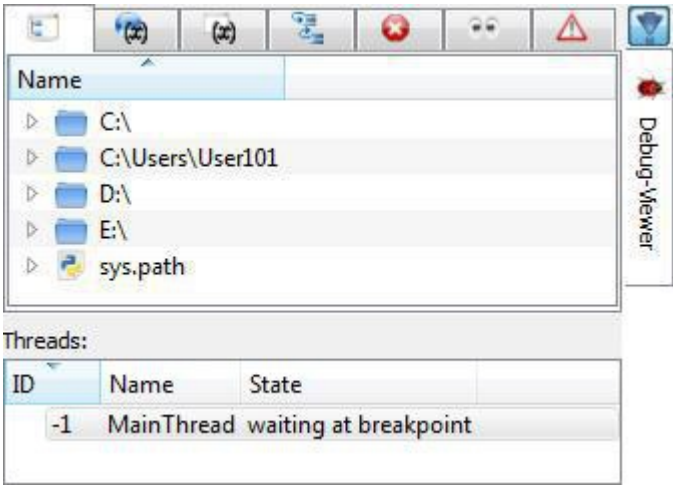
Optional in the sense that its presence can be controlled with the main menu command `Window > Right Sidebar`, selected with `Window > Edit Profile / Debug Profile` commands and configured on the "`Configure View Profiles`" control box of command `Settings > View Profiles...` [*see*].

Plus, when in display, also with the usual toggle-control button for the automatic collapsing mechanism.

– –

99 "R-Pane" hereafter, for short.

R-Pane: **Debug-Viewer**



This the `Debug-Viewer` tab, comprising two forms:

`Debug-Viewer` A tabbed form, comprising the following tabs, ordered from left to right: `File-Browser`, `Global Variables`, `Local Variables`, `Call Trace`, `Breakpoints`, `Watchpoints`, `Exceptions`; hereafter described in detail.

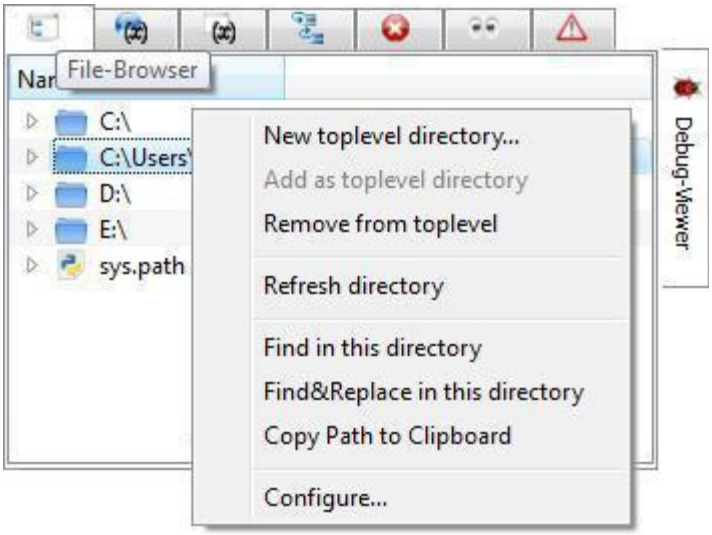
`Threads` A table meant primarily for debugging Python multithreaded programs written making use of the "`threading`" Python module[100], but shown also with the debugging of single thread programs.

– –

100An advanced programming technique about which nothing we can say, as we have matured no related experience [*ref. URL:* `http://pymotw.com/2/threading/`].

Debug-Viewer: File-Browser

A general purposes file-browser[101], with such a Python-oriented "`sys.path`" view handy available within Eric [*see*].



Command List

`New Toplevel Directory...` `Add as Toplevel Directory` `Remove from Toplevel Refresh Directory` `Find in This Directory` `Find&Replace in This Directory` `Copy Path to Clipboard` `Configure...`

[*see context:* `Tab (^)`, *in:* {2Central}]

– –

Remark

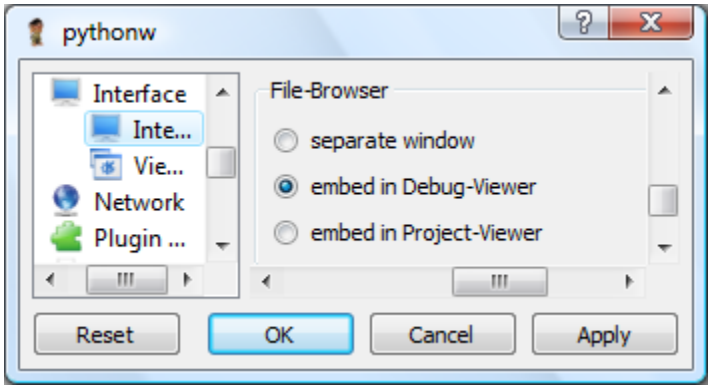
Among the possible differences between this Eric file browser and that one in use in your operating system, there is the alphabetic ordering. For instance here you'll find such a "`_XName`" not before but *after* a "`Xname`".

– –

101 Whereas the not-so-different Project-Viewer's "`Sources`" browser, on the L-Pane [*see*], is there to offer a dedicated view on the current Project.

Alternative “File-Browser” Location

It's here worth noting that, as with other Eric controls [*cf.:* Alternative Text “List View” Form, *in:* {2Central}], also this File-Browser could be configured so to be located elsewhere. For instance into the L-Pane, via command: `Settings > Preferences... - Interface > Interface, Configure User Interface, File-Browser` [*see*].



With such possibly new condition that will be then enabled at next application startup.

Viewpoint

<!> Note that also in this case throughout this Report we'll make reference to the initial default condition only, for obvious reasons of simplicity.


Besides, we had news that current File-Browser's default position “`embed in Debug-Viewer`” [*see*], in future could be moved into the “`Project-Viewer`”, on the L-Pane. Well, just an example of a minor possible change, and consequent discrepancy from this Report, that any user should be prepared to face and overcome with no difficulty.

– –

File-Browser (^) New Toplevel Directory...

Designed to bring here in display[102], as “Toplevel Directory”, another directory of your file system, as for instance a: `C:\Python33\Lib\site-packages\eric5`; so to get an easy first-hand look into it. Inverse action with next `(^) Remove from Toplevel` command [*see*]. Original directory otherwise unaffected.

102That is: no creation is here involved, as the *New* keyword could lead you to assume.

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.

– –

File-Browser (^) Remove from Toplevel

Designed to remove a "Toplevel" display condition of a directory, that is 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.

– –

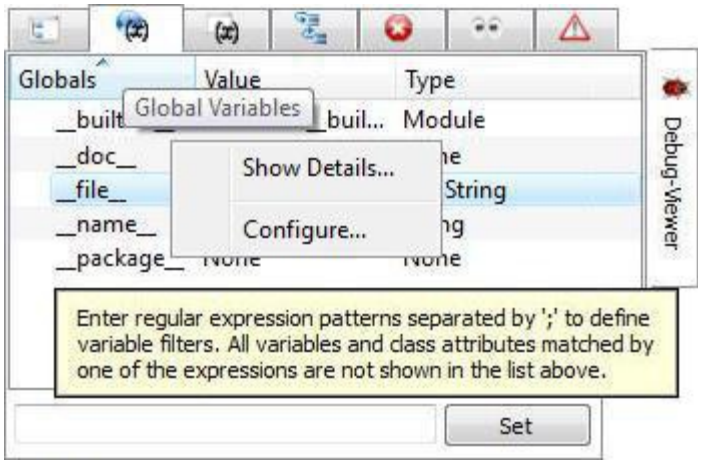
File-Browser (^) Configure...

Designed to show the same `Settings > Preferences...` window [*see*], but conveniently opened on the "`Configure general debugger settings`" control box.

-<>-

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?`]

– –

Command List

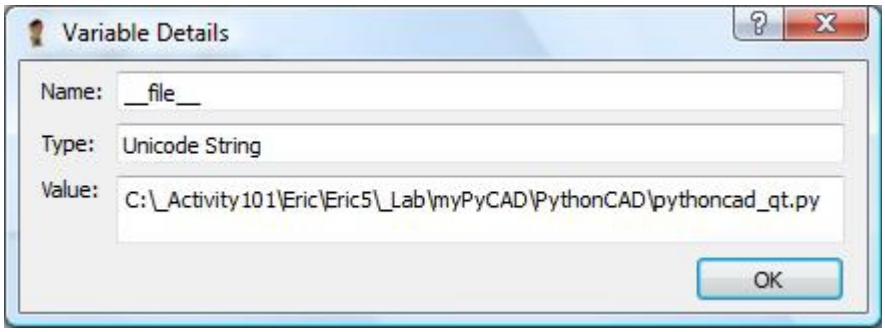
`Show Details...`

`Configure...` [*see context:* `File-Browser (^) Configure...`]

– –

Global Variables (^) **Show Details...**

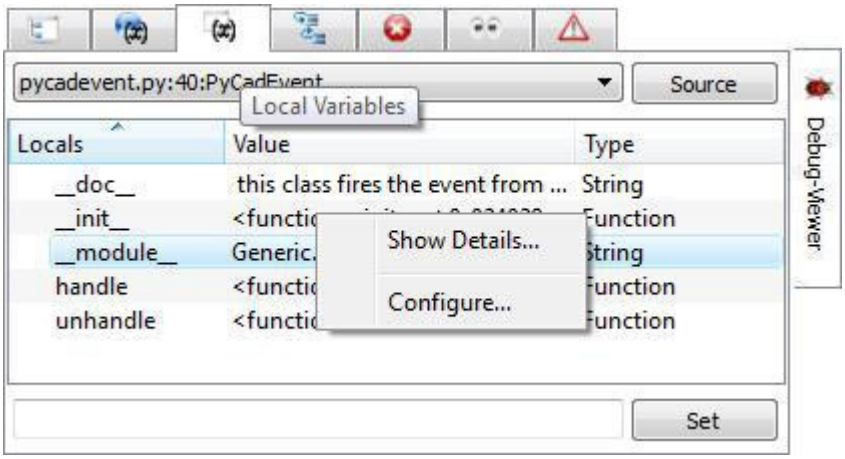
Designed to show a "`Variable Details`" dialog box where to display all details of the selected Global Variable.



A double-clik on that variable will have the same result.

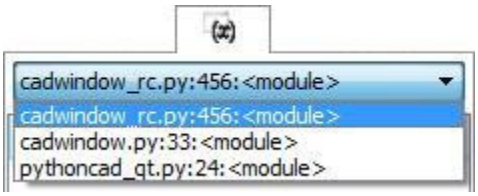
-<>-

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 [*see*].

`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*].

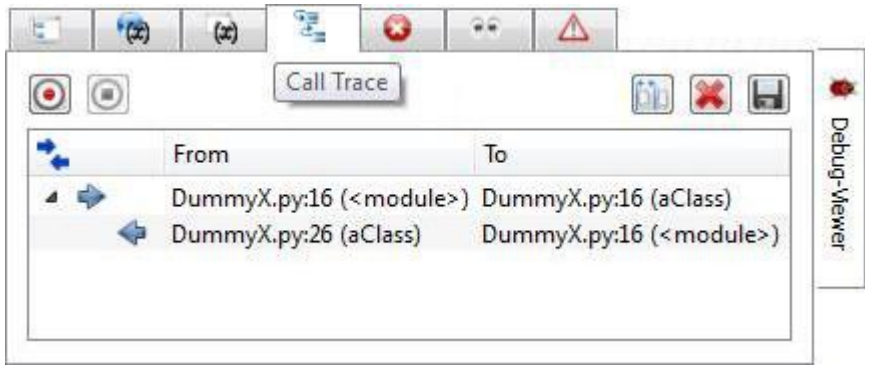
Command List

`Show Details...` [*see context:* `Global Variables (^) Show Details...`]
`Configure...` [*see context:* `File-Browser (^) Configure...`]

-<>-

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:

 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

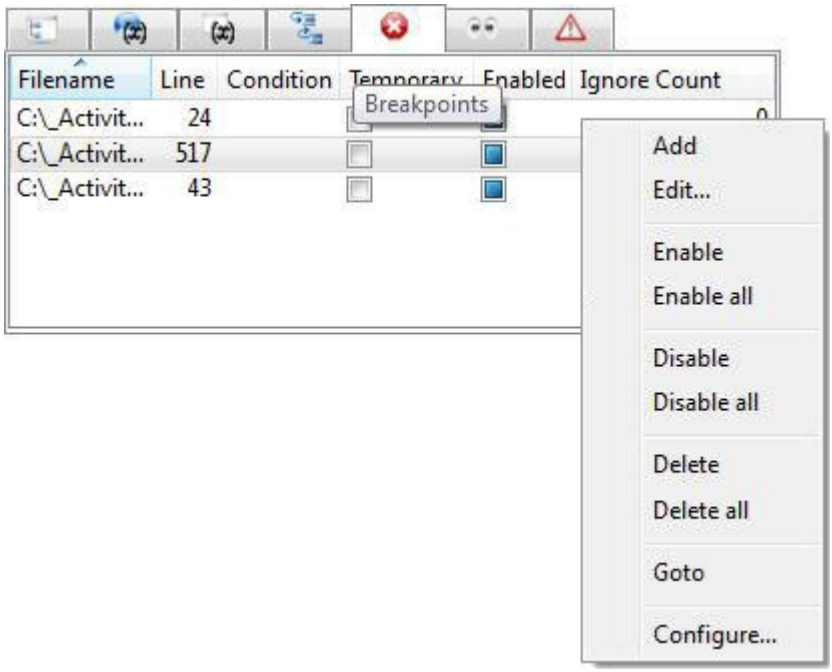
 To reset and clear tracing data

To save current tracing into a text file[103]

All actions performed through these controls, no context menu needed.

-<>-

Debug-Viewer: **Breakpoints**



Command List

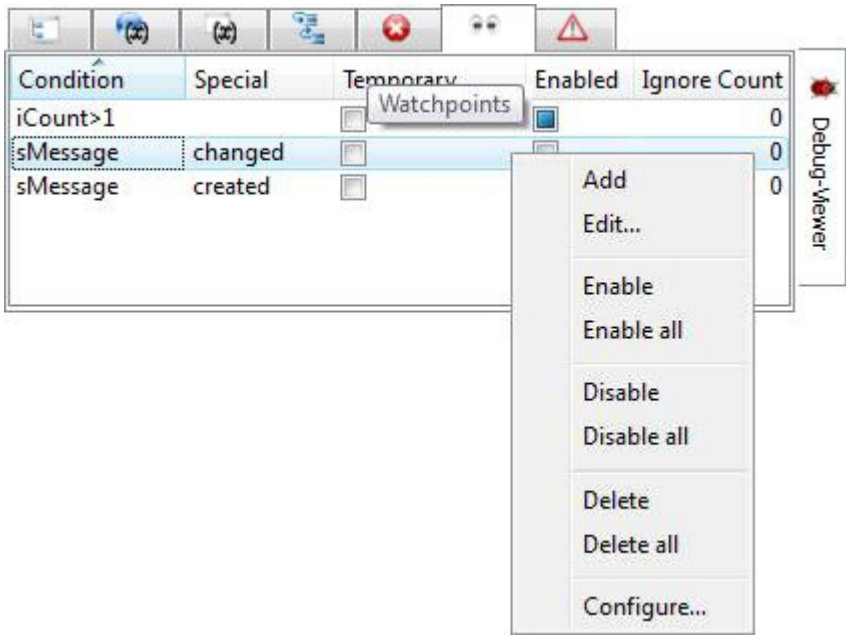
```
Add          Edit...                    [see menu: Debug > Toggle, Edit … Breakpoint]
Enable       Enable All                 [see menu: Debug > Toggle, Edit … Breakpoint]
Disable      Disable All                [see menu: Debug > Toggle, Edit … Breakpoint]
Delete       Delete All       Goto      [see menu: Debug > Toggle, Edit … Breakpoint]
Configure...                              [see context: File-Browser (^) Configure...]
```

-<>-

103<~> We'd suggest a better choice for the default directory for this file. That is the Project's, instead of the Python's installation directory.

Debug-Viewer: **Watchpoints**

Eric Watchpoints, an advanced form of breakpoint bound to an execution condition, instead of simply a source code line. They operate during debug execution and are to be defined and managed by means of the hereafter context commands [*in particular:* `Add` *and* `Edit...`].



Command List

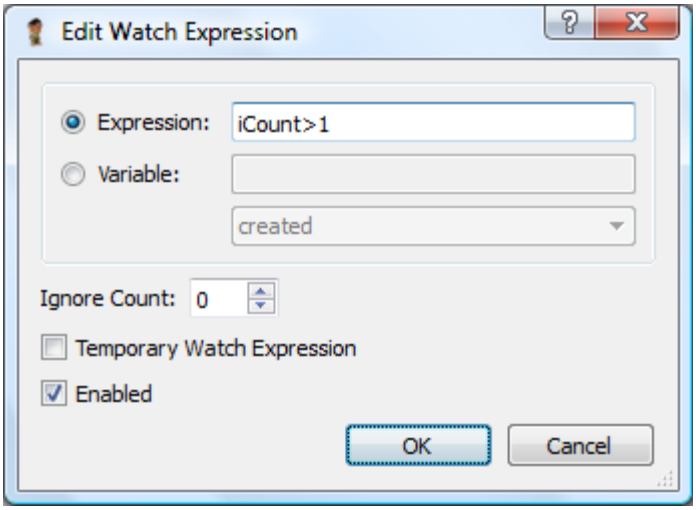
`Add` `Edit...`
`Enable` `Enable All` `Disable` `Disable All` `Delete` `Delete All` [*self-explanatory*]
`Configure...` [*see context:* `File-Browser (^) Configure...`]

– –

Watchpoints (^) **Add**

Watchpoints (^) **Edit...**

Designed to show this "`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` which is 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` Two special "watch" events for a given variable, that is when: `created` / `changed` A condition in alternative with former "`Expression`" [*see*].

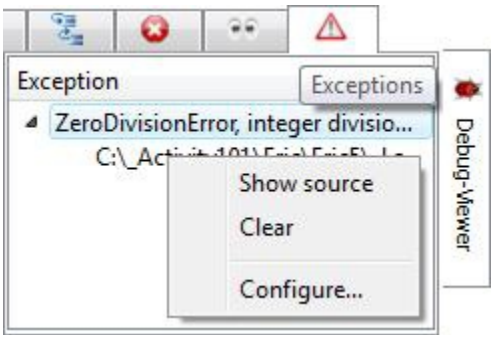
`Ignore Count` Condition to be ignored that many times, typically useful in debugging execution loops [*cf.:* `Debug > Edit Breakpoint...`]

`Temporary` Condition to be executed just one time, then disabled [*cf.:* `Debug > Edit Breakpoint...`]

`Enabled` To enable / disable a condition

-<>-

Debug-Viewer: **Exceptions**



Command List

`Show Source` `Clear`
`Configure...` [*see context:* `File-Browser (^) Configure...`]

– –

Exceptions (^) **Show Source**

Designed to synchronize the item selected on the `Exceptions` list with the related source text form.

– –

Exceptions (^) **Clear**

Designed to clear all items possibly listed on the `Exceptions` form.

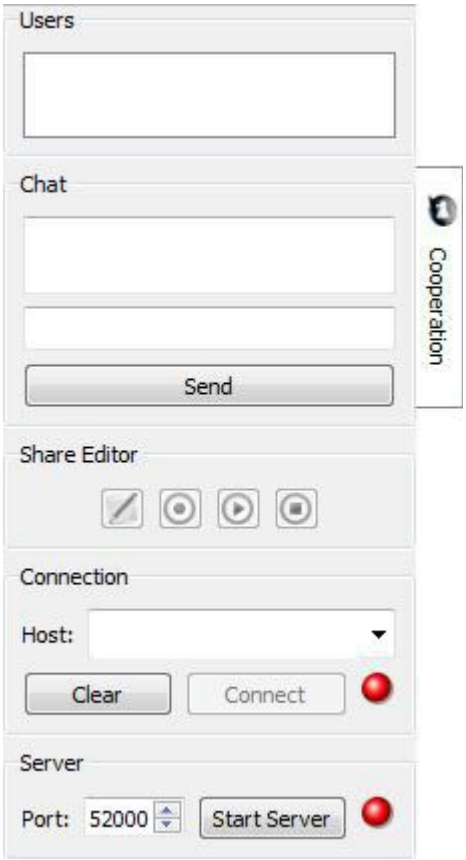
-<>-

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 end, 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"[104] 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[105] called: *well-known ports* 0÷1023, *registered ports* 1024÷49151, and *dynamic* or *private ports* 49152÷65535[106].

Hereafter an example of two Eric instances cooperating on the same computer.

104"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 you have to tell your counterpart what is the external IP-Address of your router.

105A convention overseen by the Internet Assigned Numbers Authority (IANA).

106<~> Note that the initial Port number 52000, here suggested as default, is prudentially chosen not within the set of *registered*, but within that of *private*, typically used as "ephemeral" ports [*see:* `Settings > Preferences... – Cooperation, Server Port`].

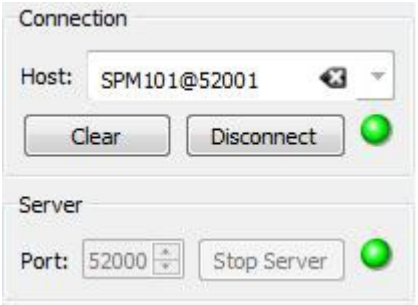
Note that the Port number chosen for the second Eric instance on the same computer is automatically proposed different from the first one.

– –

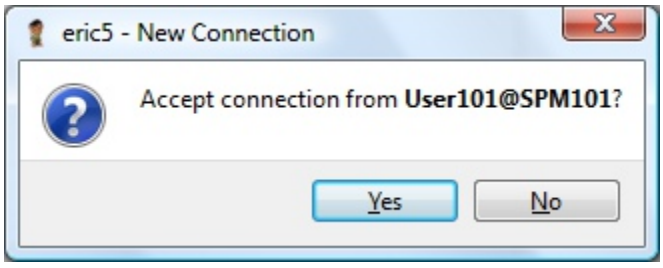
`Host` A [<~> *rather misleading*] label for the "`Host@Port`" parameter to be entered here, so to cooperatively connect the current Eric instance to another cooperating instance. Specifications:

`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[107], such as e-mail or voice.

– –

`Users` It's a 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[108].

107<~> Not a negligible trouble, we'd dare say.

108<~> Anyhow here we doubt that *any* of the participant be actually enabled to "Kick" or "Ban" *any others* [*see*].

- Connected nodes are listed on the "`Users`" area with an identifying code which has got a [*curiously*] different format from the "`Host@Port`" as required to activate a connection [*cf. above "`Host`" fields*].

Viewpoint

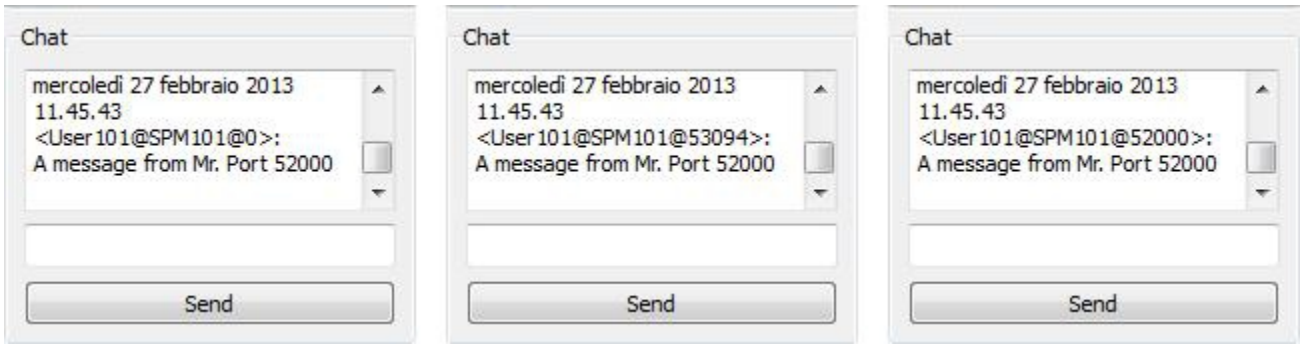
<~> "[*curiously*]" is our (mild) expression of perplexity that we'd like to justify with this example:

- 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 Mr. `User101@SPM101`
- And, 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 with the same format. We are fairly sure that all these different ways for identifying precisely the same subject have some solid justification, nevertheless we can't help expressing our (mild) perplexity about so different formats used for referring to the same thing.

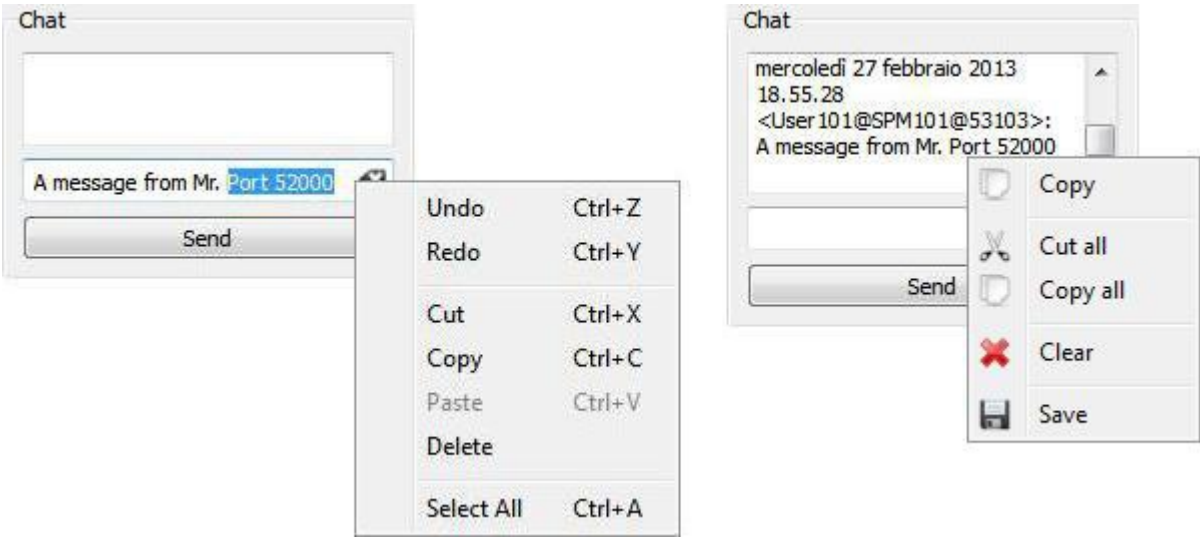
– –

`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 characters.
- 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 Network community of Eric users.

Once connected to the Internet, IRC *Servers* are there available to 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 Eric people. See this  button[109] to manage (add / edit) the set of IRC Networks here available.

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 [*see*].

– –

To connect / disconnect the selected Network.

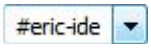
Once connected, here is where the Network messages are shown:

– –

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.

– –

109Detailed description of this feature is here considered beyond the scope of this Report [*see:* Scope of this Report, *in:* {0Lead}].

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[110].
- 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.

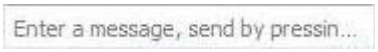
– –

Buttons to join / leave current Channel.

Once joined, here is where the Channel messages are shown:

110<~> And, frankly, as we'd expect.

– –



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 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.

– = –

{6Trail} Setup and General Management

Reference section for setup and general management of Eric (5) Python IDE, onto the target environment here considered, that is: host operating system Windows Vista Ultimate, or Windows XP Pro [*see:* Essentials, *table* Version Reference, *in:* {0Lead}].

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's convenient to subdivide and classify all such packages into *Required* and *Optional*.

– –

Required Packages

1/2] Python 3 Main Programming Language, current version [*cf. item:* Python 2, *in next section:* Optional Packages]

2/2] PyQt4 GUI Library

Eric (5) Chief application

Related operative details on subsequent sections, ordered in the precise installation sequence as required by Eric.

– –

Optional Packages

Python 2 Same programming language, vintage version, Eric (5) compatible [*see:* Essentials, Python 2 – 3 Compatibility, *in:* {0Lead}]

Ruby Other Eric (5) compatible programming language

PyEnchant Spell-checking library for Python

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

– –

"Mutatis mutandis"

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

– –

Required Packages

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

1/2] Python 3

S-PM 130600

Programming language Python 3, the main toolkit here needed first.

Where

Available for downloading from URL: `http://www.python.org/download/releases/3.3.2/`

What

Download file: •`Windows x86 MSI Installer (3.3.2) (sig)` [*or more recent*]

That is, among the various available items, the latest Python 3.*x.y*, Windows x86, MSI Installer — Windows binary only, no source included[111].

How

Setup-run, with `Administrator` permission. All defaults accepted, and you'll get such a: "`\Python3`*x*"[112] directory installed [*see*].

All items on the resulting `Start > Python` button are of immediate comprehension, but perhaps the "`Module Docs`" 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.

111Then if you wander what the cryptic "`(sig)`" stands for, it's short for "*PGP signature*", as available at the bottom of the same web page, so to possibly verify the integrity of the downloaded file.

112Actually a "`\Python33`", in our current case.

Uninstall

Under the resulting system `Start > Python` button there is also a reassuring "`Uninstall Python`" command. We've prudentially executed it from here before each successive upgrading.

– –

2/2] PyQt4

S-PM 130600

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 section, where you'll find, mixed up together, material aimed at the PyQt development along with ready-to-use material. Fortunately then things are much simpler that one might expect as, at least with the Windows platform, the ready-to-use package includes also other tools listed by Eric as distinct pre-requisites[113]*. Which they are not, in the sense that you don't need to tell the difference and to look for each one of them*[114]*, as they are already part of this very same PyQt package.*

– –

Where

PyQt can be found at URL:
`http://www.riverbankcomputing.co.uk/software/pyqt/download`

What

What you need here [*see:* Essentials*, table* Version Reference*, in:* {0Lead}] is simply the Windows installer: `PyQt4-4.10.2-gpl-Py3.3-Qt4.8.4-x32.exe` [*or more recent*][115]

As for: GPL - General Public License, Python 3.*x*, Qt 4.*x*, 32 bit. A few minutes of downloading.

– –

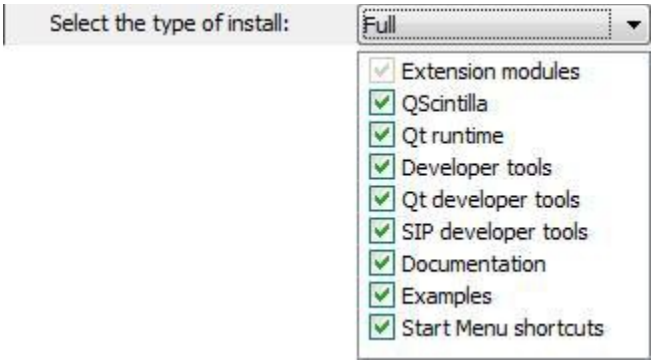
113[*cf. URL:* `http://eric-ide.python-projects.org/eric-download.html`]

114That 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.

115Defined as: "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/
<~> A not-so-clear situation that we anyhow accept "AsIs", without committing into any further investigation.

How

A setup-run, with `Administrator` permission, all defaults accepted, and you'll get a: "`\PyQt4`" 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 v4.10.2 for Python v3.3 (x32)
```

That's a folder rich with programs, `doc`[116], `examples`, … So rich that, without a proper "primer" documentation[117], it's easy to get lost.

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

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

If accepted, 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 successive upgrading.

-<>-

116A nice surprise, after having being warned that "Qt documentation is not included" [*cf. former footnote*]. Evidently there is some difference between Qt and PyQt documentation.

<~> Another not-so-clear situation that we anyhow accept "AsIs", without committing into any further investigation.

117Which we haven't been able to find out, so far. Alas.

Eric (5)

S-PM 130600

Here it's about the Eric toolkit, final and chief subject of all this setup sequence. More precisely it's about Eric ver. 5, as an IDE (Integrated Development Environment) for Python ver. 3.*x*.

Where

Eric (5) can be found at URL:
http://sourceforge.net/projects/eric-ide/files/eric5/stable/

What

Just download: `eric5-5.3.5.zip` [*or more recent*]

Remark

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

`eric5-<x>.zip`
`eric5-<x>.tar.gz`
...
`eric5-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; 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[118]. A package regarded as a functional "black-box", exclusively from a user's point of view.

– –

118Anyhow at least a look inside this package, and its sub-directory `\eric`, is highly advisable.

How

Just expand the "`.zip`" package, then locate and run the "`install.py`" procedure inside[119]. This way you'll get a rich set of such "`eric5.bat`" commands into the same Python installation directory, and a directory "`\eric5`" added into the "`\Lib\site-packages`" [*see*].

Remark

Eric (5) "`install.py`" procedure is meant to be executed by Python ver. 3.*x*. That means that, in case you've got installed into the same computer both Python ver. 2.x and Python ver 3.x—a rather convenient condition, for many reasons [*see:* Essentials, Python 2 – 3 Compatibility, *in:* {0Lead}]—to run the "right" interpreter you must explicitly specify:

```
C:\Python33\Python.exe install.py
```

So to avoid such an error condition, caused by the Windows operating system possibly picking up automatically the "wrong" interpreter Python ver. 2.*x*, instead of the required Python ver. 3.*x*[120].

Viewpoint

<~> But it's our opinion that the most elegant way for eliminating such a (minor) problem would be simply to make good use of the dedicated "`*.py2`" and "`*.py3`" source file extensions, in place of the generic "`*.py`".

In fact, as we've successfully done in our workstations, you just need to properly specify: `install.py3` and `uninstall.py3` where needed (e.g.: the "`eric5-<x>`" setup package), and then inform the Windows operating system which is the Python interpreter to be associated to that "`py3`" extension (e.g.: `C:\Python33\python.exe`).

– –

119A "`>Python install.py -h`" is to display some help [*see*].

120Other possible actions to grant the desired precedence of Python ver. 3.x over Python ver. 2.x is to install first the latter, and then the former; or to manually modify the related `Path` environment variable. We dislike both.

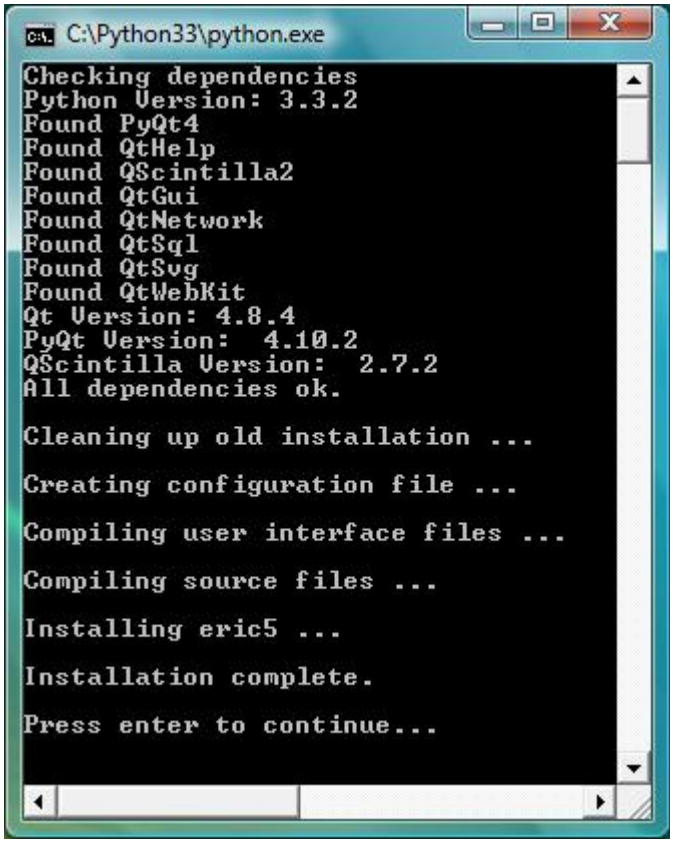
Viewpoint

<!> But our deeply rooted opinion is quite another. Such a problem shouldn't simply exist, as a fairly conceived professional tool, such as Python pretends to be, should evolve without breaking compatibility with just the previous version number.

Obsolete statements can 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 2 vs. Python 3 antagonism, and mess. Indeed.

– –

Then, this a typical installation log:



Where you're reassured about "`All dependencies ok`".

That is, in this case:

```
1/4]  Python 3.3.2
2/4]  Qt 4.8.4
3/4]  PyQt 4.10.2
4/4]  QScintilla 2.7.2
```

– –

A list showing that the toolkit `PyQt`, being already installed as a prerequisite [*see section: 2/2]* PyQt4], brings along also the toolkits `Qt` and `QScintilla`, which, clearly, don't need to be handled individually.

Then nothing new will appear on the system `Start` button, so that it is advisable to add there a custom shortcut for running Eric, as described in the next section ***Then*** [*see*].

– –

Then

At "`install.py`" completion, a test run of the "`eric5.bat`" command made available into the root Python directory could be used to verify that everything is working properly [*see*].

Remark

All the other similar "`*.bat`" commands, such as "`eric5-api.bat`", ..., which can be spotted into the same directory along with "`eric5.bat`", are meant to run various Eric tools autonomously. For instance "`eric5-webbrowser.bat`" is to run the Eric Web Browser independently from Eric [*see*].

– –

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 the Python 3 vs. Python 2 running precedence issue when executing the Eric (5)* `install.py` *procedure*]. No message at execution completion[121].

In case of an Eric upgrading, the `install.py` procedure can be called directly—that is: without a preceding `uninstall.py`—as its preliminary action will be a "`Cleaning up old installation...`" [*see the installation log image, reproduced here above*].

– –

121<~> Frankly here we'd appreciate a final "`Uninstall done ok`" message.

Viewpoint

In general, and in this case too, the very concept of "uninstall" should deserve a better definition. It is our strong opinion that a fair uninstall process should erase *every aspect* of the installed stuff, leaving the host system precisely as it was before the corresponding installation. Clearly and neatly. That is to say, just for one thing, 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?

More precisely we could make a difference between a sheer-uninstall, that is when you really want to "get rid" of an installation, and an "upgrading"-uninstall; that is when keeping track of, say, such elements as: configuration parameters, file history and so, is required and welcome.

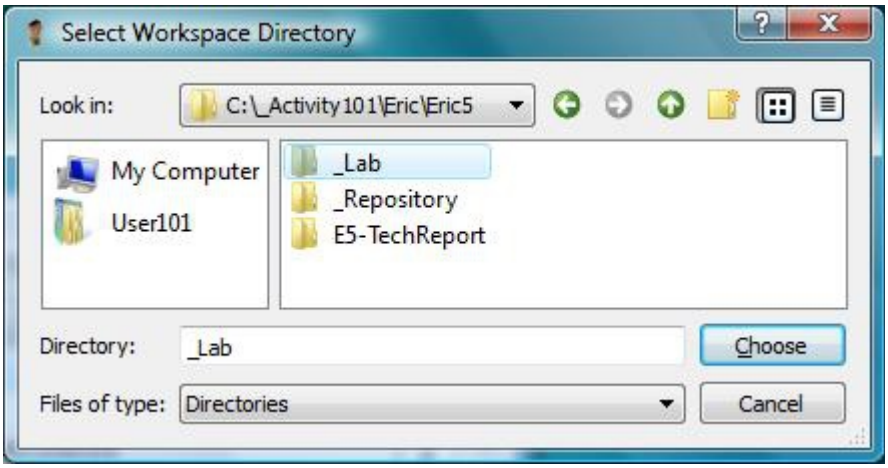
<~> Here, for instance, with Eric this difference could be granted by a conveniently different behavior between the very "`uninstall.py`" procedure and the uninstall function implicit into the "`install.py`" procedure[122].

-<>-

122By the way, it's why, when upgrading Eric, we almost ritually execute first and "independent uninstall", instead of an "install over install" procedure. Anyhow, even though we know that currently there is no difference. It's because we love so much fresh restarts.

Eric First Run

At first Eric run you'll be asked to select an initial default value for the Eric Workspace directory:

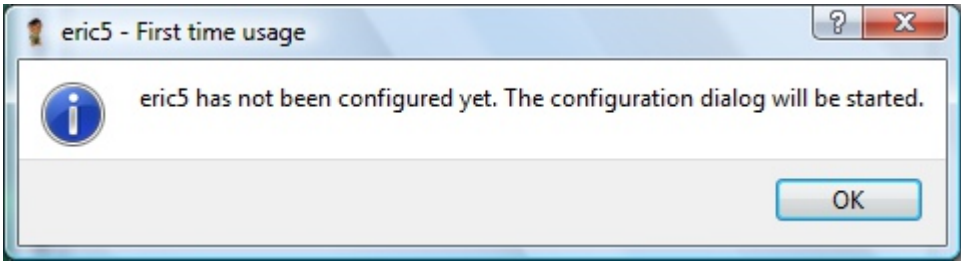


A value then changeable at any moment on the `Workspace` field of: `Settings > Preferences... - Project > Multiproject, Configure multiproject settings` [*see*].

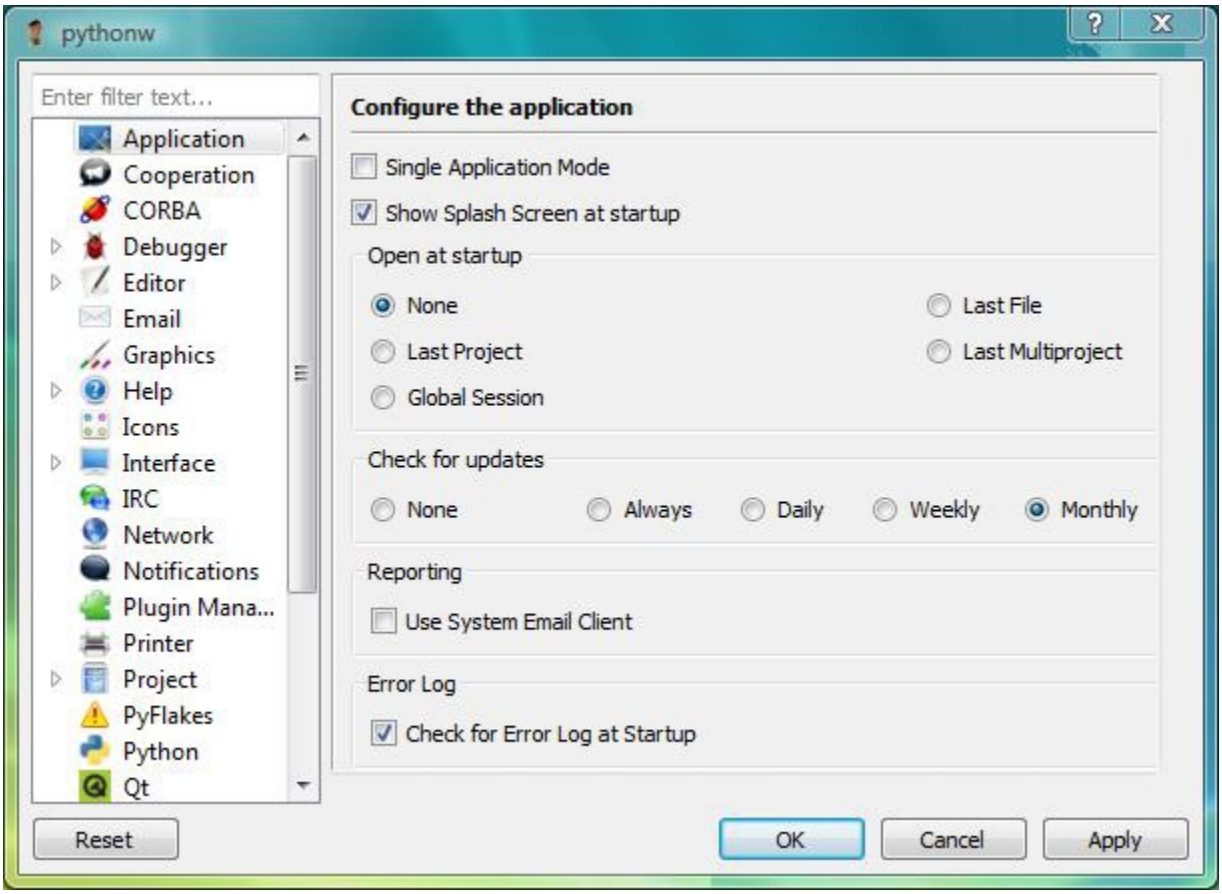
On subsequent runs on the same station, even after an Eric upgrading, you'll be not asked again, as all configuration data are kept.

– –

Anyhow, in case of such "`First time usage`" box:



you will be requested to enter a click action, so "`to display the configuration page`":



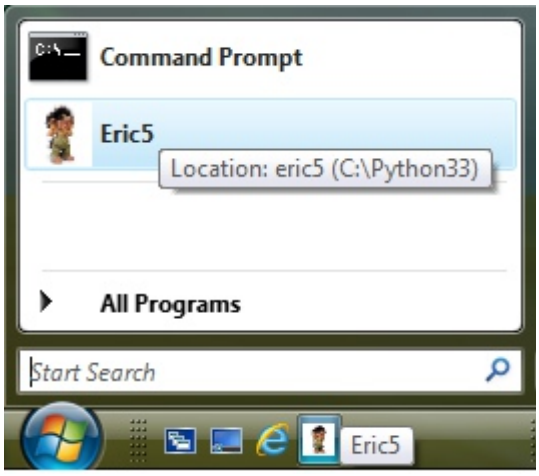
In any case this same condition can be brought up also running independently the `eric5_configure.bat` command macro, from the root Python directory, or also at any other moment during a usual Eric session, with command: `Settings > Preferences... - Application, Configure the application`

-<>-

Custom Eric `Start` Command

<!> For running Eric it is convenient to create into the Windows system `Start` button a shortcut referring to the standard "`eric5.bat`" command[123] as it is available into the Python root directory [*see*].

Here is how to create such a useful shortcut:



1] Locate the `eric5.bat` command into the `C:\Python3x\` directory;

2] Right-click it to "`Create Shortcut`", and keep this shortcut there;

2.1] Into the `\eric5\pixmaps` directory there is a nice `eric5.ico` that you could assign to it;

3] A drag-drop copy of such shortcut can be created into the system `Start` menu, possibly renaming it as "`Eric5`", or also into the system `Taskbar` [*see figure above*], or on the desktop; keeping anyhow also the original shortcut as with the former point.

– –

Remark

No instruction is here offered about how to run *separately* 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.

– –

123A look at the text of this "`.bat`" command and you'll see that it refers to a "`eric5.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—or even require—some subsequent actions of setup completion, as those hereafter listed.

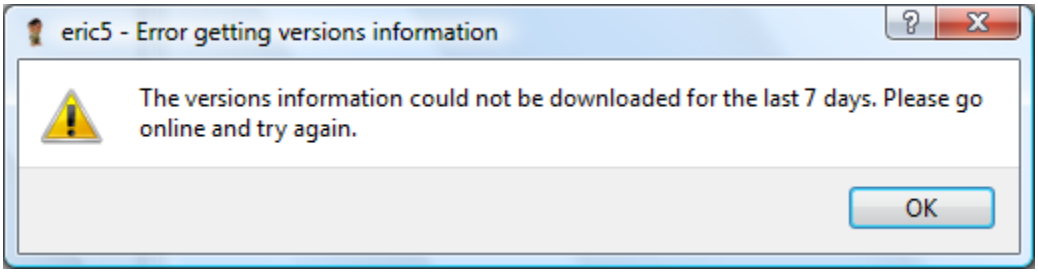
– –

Menu command: `Settings > Preferences...`

<。> On control form: `Application, Configure the application` [*see*]:

`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.

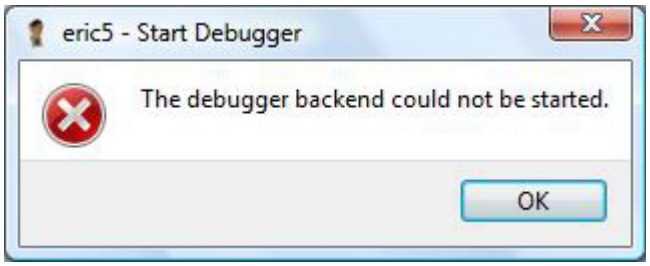
`Check for updates`
: Set of option buttons where to possibly select the option `None`, so to avoid such a unexpected warning at some Eric run:



– –

<。> On control form: `Debugger > Python[x], Configure Python[x] Debugger` [*see*]:

`Python[x] Interpreter for Debug Client`
: That is the area to inspect, and “fix”, in case such an error box should appear:



– –

<。> On control form: `Editor > Filehandling, Configure file handling settings` [*see*]:

`Default File Filters`

Drop-down lists, where to conveniently set default file extensions, as for instance:

```
Open files:  *.py *.py2 *.py3
Save File:  *.py
```

Being the initial setup values a not very convenient "blank"[124].

– –

<。> On control form: `Editor > Style, Configure editor styles` [*see*]:

`Fonts` Check-box where to possibly set: `Use monospace as default`

– –

<。> On control form: `Help > Help Documentation, Configure help documentation` [*see*]:

`Python` and `PyQt4 Documentation`[125]

Text fields where to set such paths as:

```
C:\Python33\Doc\python332.chm
C:\Python33\Lib\site-packages\PyQt4\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 [*see*]; manageable via DOS "`set`" command[126], or Python "`os.environ`" dictionary.

– –

124Implying as default simply the first item in the list, such as: `Batch Files (*.bat *.cmd)`.

125<~> Sorry for not having any suggestions for Qt4, Qt5.

126We have to confess that, after a first hasty and unsuccessful attempt, we've renounced exploring this possibility.

<。> On control form: `Help > Help Viewers, Configure help viewers` [*see*]:

`Help Viewer`

A set of option buttons, where to possibly select: `Eric Web Browser`

– –

Viewpoint

To the benefit of whom do not appreciate some extreme Gothic aesthetic expressions, and would like to gain control over the appearance of the initial splash-screen and other Eric images, we inform that this is the directory where to find them:

`C:\Python3`*n*`\Lib\site-packages\eric5\pixmaps`

Where you'll find: `ericSplash.png`, `eric_small.png` and `eric.png`, rather easily editable the way they are shown throughout this Report.

Alternatively you may have them even not shown, this way:

- Setting off the "`Show Splash Screen at startup`" check box, on: `Settings > Preferences... – Application, Configure the application`
- Modifying such file name as: `eric_small.png` to, say: `-eric_small.png`, so to "hide" it altogether from Eric. With no error condition generated, as we've verified.

-<>-

Optional Packages

Eric optional packages can be installed at wish, at any moment, in no special order. What here follows are some real-case, notable examples.

`Python ver. 2` Can be installed beside the pre-requisite Python ver. 3, so to actually enjoy what declared in section "Python 2 – 3 Compatibility" [*see in:* {0Lead}]. For technical details refer to former "*1/2]* Python 3" section, changing what is to be changed[127].

`PyQt4-(Py2)` PyQt companion GUI library for Python 2 [*cf. above section:* *2/2]* PyQt4].

`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:* {1North}].

`Mercurial` and `Subversion` Two well known Version Control Systems (VCS)[128], here considered with menu command `Project > Version Control` [*see in:* {1North}].

`PyEnchant` Spell-checker toolkit, here considered as a notable example with command `Settings > Show External Tools` [*see in:* {1North}], required by menu command `Extras > Check Spelling...`, and also by "Check Spell Menu" context menu [*see in:* {2Central}].

– –

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.

– = –

127Or the other Report about Eric 4, entirely based upon Python 2, as available at the Eric web site [*see:* Foreword, *What & Where in Internet*].

128<。> Specifically about PySvn Subversion for Eric 4, a Report has already been produced, available as with preceding footnote. Mercurial is assumed as the main VCS for Eric 5.

Appendix

Sections and Revision

This whole Report is composed of distinct "*Sections*", delimited by such Begin-Of-Section (BOS):

S-PM 130500

and such End-Of-Section (EOS) lines:

– –

<!> 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[129] is assumed to play a relevant role on the future life of this Report.

– –

Top-Sections Prefix Code

For all Top-Sections has been introduced such a "{*nTag*}" prefix code, reproduced on all page headers as a convenient reference and navigation tool:

{0Lead}	Intro
{1North}	North Side, on the Eric Application Window
{2Central}	Central Park
{3South}	South Side
{4West}	West Side
{5East}	East Side
{6Trail}	Setup & Management
{Map}	Map of the Eric Application Window

– –

129 Besides the usual disclaimer telling: "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 specify 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: URL `http://sourceforge.net/projects/` File path `C:\Program Files\` Menu command `Settings > Preferences...`
`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 > 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: <!> Remarkable point <?> Questionable point <~> Perplexity, doubt, suggestion <...> Topic still to be completed <.?.> Questionable point, still under investigation
	Icon for an "*Eric Feature Under Revision*", whose Tech.Report is currently left AsIs.

-<>-

Reference Book List

S-PM 130500

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[130], 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:* {1North}]

Python Essential Reference
: by David M. Beazley, Addison-Wesley

PyQt 4 Reference Guide
: Riverbank Computing Ltd, as with the Eric `Help` command [*see in:* {1North}]

Rapid GUI Programming with Python and Qt
: by Mark Summerfield, Qtrac Ltd

- = -

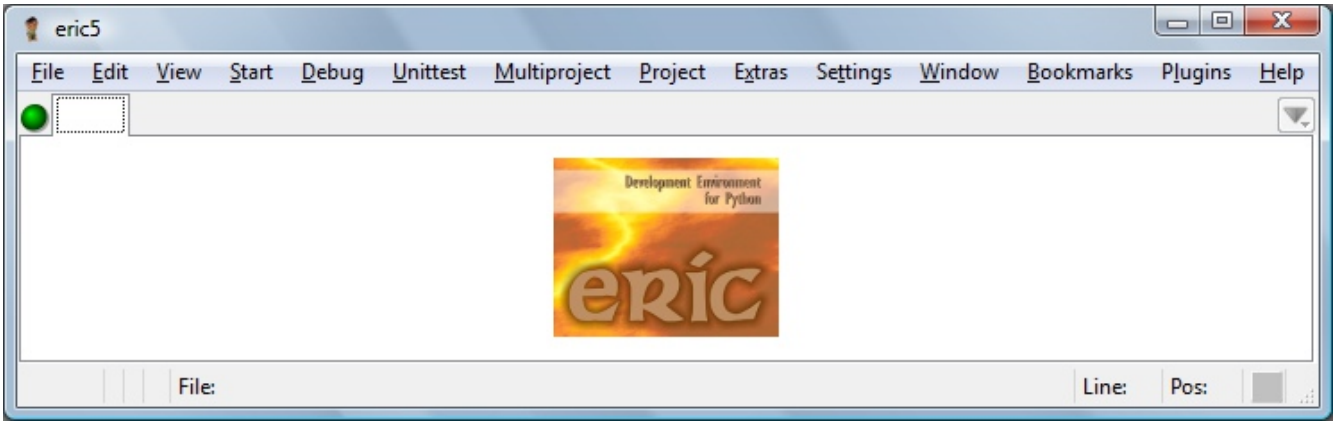
130And, why not, also the other Reports as available on the Eric web site [*see:* Foreword, *What & Where in Internet*].

Table of Contents

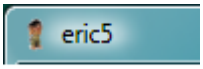
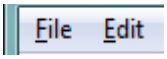
– = –

{Map} Eric Window Map
and Glossary

Maps of the Eric (5) application window, aimed at locating all graphic objects, listed along with their standard denomination and the section of this Report where they are specifically treated.

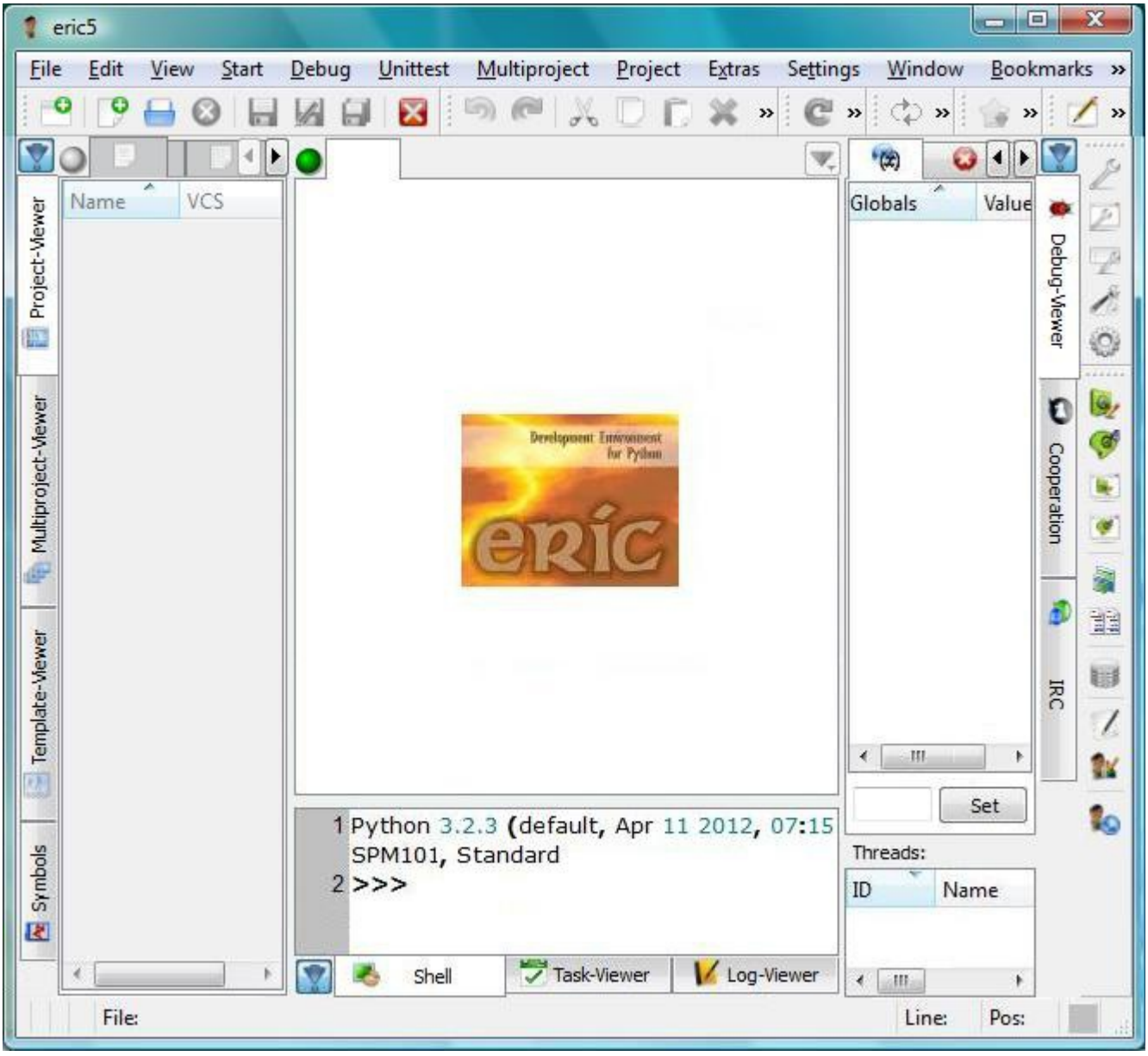


This the Eric Window in its simplest appearance, corresponding to the main menu command `Window > Edit Profile`, with all related items un-checked [*see*]. Configured this way, the work area hosts exclusively one form, the *Central*, that is the Tagged Text Form, where Python source modules are edited. None of the Auxiliary Forms—the Left, Bottom and Right [*see next map*]—are shown.

Graphic Item	*Denomination*	*Ref. Section*
	Title Bar	{1North}
	Menu Bar	{1North}
	Text Edit Form	{2Central}
	Information Bar	{3South}

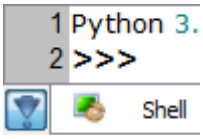
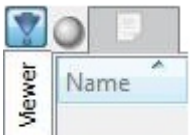
-<>-

Here is the usual Eric Window, in its initial default appearance[131], where to locate all the main graphic objects not shown on the preceding, simpler, map [*see*]. That is:



Auxiliary *Left*, *Bottom*, and *Right Pane*, located around the stable *Central Pane* on the application work area; plus the horizontal and vertical *Tool Bars*.

131Here just slightly modified, so to better fit it into this page.

Graphic Item	*Denomination*	*Ref. Section*
	Tool Bars	{1North}
	Bottom Pane	{3South}
	Left Pane	{4West}
	Right Pane	{5East}

-<>-

Glossary

S-PM 130700

Glossary of terms as used throughout this Report, referring to items on this Eric application window.

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 A secondary Window for specific user interactions. Likewise, just a bit more specific: *Control*, *Error*, *Tool*, *Alarm*, *Hint Box*; or simply: *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

Source Text Form

Or Text Edit Form, or simply Text Form: the Main Form of this application, where source text is edited[132]. With this 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.

– = –

132So that: a "*Text Editor*" is the tool you use to write into the "*Text Form*".