About ConTEXT

ConTEXT is small, fast and powerful text editor developed to serve to software developers.

This editor is freeware, absolutely free for use. If you are so fascinated and want to pay for it, I encourage you to send any amount of anything to the address listed at the bottom of this file. Anyway, I'd like to hear any of your comments and suggestions, to discuss it and implement it in future versions.

Check ConTEXT pages at http://www.fixedsys.com/context for ConTEXT updates. New versions with added features and bugfixes are available very often.

Thanks to Germán Giraldo for providing me basic help file.

What's new

See History.txt file located in your ConTEXT directory.

Planned Features

* enhancing macro recorder features and macro script language
* code browser for C/C++, Delphi and Visual Basic projects
* file compare
* more highlighters
* paragraphs and real word wrapping
* other misc. tools
* complete help file

Contact

Postal address:

Eden Kirin
Remetinecka 151
10020 Zagreb
Croatia, Europe

E-mail:

foetus@aphex.fpz.hr

Mailing list:

see details on support pages

Home page

http://www.fixedsys.com/context

File

New...
Opens a dialog box to create a new file.

Open...
Opens a dialog box to open an existing file.

Close
Close the currently open file.

Save
Save the current file. This command is available when change to your file have been made.

Save As...
Saves the current file under different file name.

Save All Files
Save all open files..

Copy To
Copy current editing file to the new file without renaming it in editor.

Rename...
Opens a dialog box to rename current file to new file name.

Print
Opens a dialog box to print current file, with the options defined in Page Setup.

Page Setup...
Opens a dialog box to define options to print. See Page Setup Dialog Box

Print Preview...
Opens a window to view the current file in screen before you print it. See Print Preview Window.

Insert File At Current Position...
Opens a dialog box to select file to insert at current cursor position.

Append File...
Opens a dialog box to select file to append at end of current file

Write Block to File...
Opens a dialog box "Save as" to enter file name where it's saved the text selected. This command is available when there is text selected.

Export
Export current file in HTML or RTF format, to file or clipboard.

Close
Close current open file. If change to your file have been make, will show a message box requesting if you want to save the changes.

Close All Files
Close all open file in a similar form to Close.

Recent Files
List of recently used files. Select one if you want open it.

Recent Projects
List of recently used projects. Select one if you want open it.

Exit
Exit ConTEXT.

Edit

Undo
Reverse the last actions you made to the text.

Redo
Reverse the last actions made with Undo.

Cut
Cuts the selected text and copies it to the clipboard.

Copy
Copies the selected text to the clipboard.

Paste
Inserts the content of the clipboard into the text.

Delete
Delete selected text.

Select All
Selects the whole text at once.

Find...
Opens the Find text dialog box.

Replace...
Opens the Find and Replace dialog box.

Find Next
Find next occurrence of text entered in Find dialog box.

Find Previous
Find previous occurrence of text entered in Find dialog box.

Toggle Selection Mode
Toggle normal/column selection mode. Hold down the ALT key while making your selection with the mouse for column select text.

Sort Text
Sort selected text. Select text in columnar mode to sort it using some other column as the sort key.

Match Braces
Go to matching brace.

Select Text Between Braces
Go to matching brace and select text between braces.

View

Toolbar
Show/hide toolbar.

File Tabs
Show/hide file tabs.

Search Results
Show/hide window for Search Results.

Console Output
Show/hide window for Console Outputs.

File Explorer
Show/hide file explorer.

Set Bookmark
Show submenu to set bookmark.

Jump To Bookmark
Show submenu to jump to bookmark.

Go To Line
Opens a dialog box to enter line number.

Lock File For Changes
Display file in view-only mode.

Format

Indent Block
Indent lines containing selected text. This command is available when there is text selected.

Unindent Block
Unindent lines containing selected text. This command is available when there is text selected.

To Lower Case
Convert to lower case selected text or char to right of the cursor

To Upper Case
Convert to upper case selected text or char to right of the cursor

Invert Case
Invert case (lower to upper and upper to lower) in selected text or char to right of the cursor

Reformat Paragraph
Reformat paragraph to fit between column 1 and right margin, as defined in Environment options dialog.

Fill Block
Replace and fill columnar selected text with char entered in dialog box. This command is available if text is selected in columnar selection mode.

Insert Code From Template
Show popup list to select template code

Comment/Uncomment Code
Comment/Uncomment current line or selected text

Remove Trailing Spaces
Remove trailing spaces from current file

Project

New...
Create new project file that contains information about set of files, their current editing positions etc.

Open...
Open existing project file.

Close
Close project file. All visual parameters will be automatically saved.

Project Files
Click on project file name to open it in editor if closed.

Recent Projects
List of recently used projects. This list is identical to the one found in File menu.

Manage File List
Open dialog to add or remove files from project.

Tools

Record Macro

Start record macro. After this command, new dialog is open when you can enter macro name and shortcut key. To stop the recording, select again "record macro" or press CTRL+F8.

Play Macro

Show list to select macro. You can press directly the hot key if you don't want to see this list.

Manage Macros

Using Manage macros dialog, you can delete macro, change name or reassign hotkey of macros.

User Command (1..4)

Execute command associated with current file extension. See Environment options.

Shell Execute

Shell execute file with associated application. For example, if .html file is opened, Internet Explorer will be open with your file in it. If file is changed, it will be saved to disk before executing this command.

Set Highlighter

Program automatically select highlighter associated with current file extension. Use this command to change highlighter. You can use Customize Types dialog to change default assignment for file type extensions.

Convert Text To

Convert text to format specific to OS: DOS, UNIX, MAC.

Options

Environment options
Opens a dialog box to configure ConTEXT settings. See Environment options for more details.

Code Templates
This command opens a dialog box to edit code templates and create files if not exist.

Wordwrap
Work only when you type text.

Stay On Top
Maintain to ConTEXT on top.

Export Registry Settings
Save current settings from Windows registry to file. To restore these settings, double clicked over file name in the Windows Explorer.

Command line parameters

As ConTEXT can accept normal file names from command line, it can also accept wildcards. For long filenames, they should be enclosed in quotes "".

```
ConTEXT *.cpp c:\test\t*.php
```

This will open all cpp files in current directory and all files from c:\test that begins with t and has extension php.

Various command line switches can be used:

/prj filename.cpr

Opens ConTEXT project file

/i filename

Opens files listed in file. You may specify that the filename on the command line contains a list of files to open by using a "/i" parameter on the command line. In this case, ConTEXT will read each line of the files on the command line, and attempt to open each file. The file specified on the command line when the "/i" parameter is used must contain only filenames, and each filename must be on a separate line.

/p filename

ConTEXT will open file, print it and close it. If no other files are open, ConTEXT will be closed.

filename **/g##:##**

Opens file and jump to specified location in file. First number is column and second is line. For example:

ConTEXT myfile.txt /g10:20 will open file myfile.txt and jump to column 10, line 20.

filename **/r**

Opens file locked to be viewed only.

filename **/m "macroname"**

Opens file and runs macro on this file.

/s

This command can be used as part of macro running command. Example:

```
ConTEXT myfile.txt /m "Remove dots" /s
```

This example will open file, run "Remove dots" macro on it and save it back to disk.

/c

This command closes file and can be used as part of macro running command. Example:

```
ConTEXT myfile.txt /m "Remove dots" /s /c
```

This example will open file, run "Remove dots" macro on it, save it back to disk and close it.

/e

This command exits ConTEXT and can be used as part of macro running command. Example:

```
ConTEXT myfile.txt /m "Remove dots" /s /c /e
```

This example will open file, run "Remove dots" macro on it, save it back to disk, close it and exit ConTEXT.

Command line parameters can be combined:

```
ConTEXT c:\test.cpr /r /i myfiles.lst /g12:15 *.txt /m "My macro" /s /c /e /p file.prn
```

Compiler output parser

If rule is defined, ConTEXT will parse compiler output and try to find file name and error line number in compiler output.

Rules:

*	matching any text
%n	extracts file name
%l	extracts line number
*	*
\%	%
\\	\

Examples:

Turbo Pascal

Compiler output:

```
"C:\TEST\XBASE\XBS_LIST.PAS(7): Error 15: File not found (MYWIN.TPU)"
```

Rule:

```
"%n(%l)"
```

Turbo Assembler

Compiler output:

```
"**Error** C:\TEST\XBASE\SWAPA.ASM(212) Undefined symbol: COSTLEN"
```

Rule:

```
" *%n(%l)"
```

(space character is at the beginning of the line)

Turbo C

Compiler output:

```
"Error c:\test\x.c 110: Unable to open include file 'stdlib.h'"
```

Rule:

```
"Error %n %l:" or "* %n %l:"
```

Custom highlighters

Highlighter file is a simple ASCII text file.

You can also put comment lines by placing a two slash characters at the beginning of a line. The rest of this line will be ignored. Comments are interpreted as comments only if it placed at the beginning of the line. For example:

// This is a comment. I could put reminders to myself here...

To define custom syntax highlighter, copy ..\Highlighters\x86 Assembler.chl to new file (with your language name) and edit it. File (x86 Assembler.chl) is well commented and there should be no problems understanding it.

> Check ConTEXT support pages for additional highlighters.
>
> **Note:** Install highlighters you need. Lots of custom highlighters can slightly increase ConTEXT loading time.

// Language name (user language name)
Language: x86 Assembler

// default file filter
// note: if more than one extension is associated, eg:
// C/C++ files (*.c,*.cpp,*.h,*.hpp)|*.c;*.cpp;*.h;*.hpp
Filter: x86 Assembler files (*.asm)|*.asm

// help file which will be invokend when F1 is pressed
HelpFile:

// language case sensitivity
//   0   - no
//   1   - yes
CaseSensitive: 0

// comment type:
// LineComment - comment to the end of line
// BlockCommentBeg - block comment begin, it could be
// multiline
// BlockCommentEnd - block comment end
LineComment: ;
BlockCommentBeg:
BlockCommentEnd:

// identifier characters
// note: characters shouldn't be delimited, except arrays
// array of chars could be defined as from_char..to_char
// IdentifierBegChar - Identifier begin characters
IdentifierBegChars: a..z A..Z _%@.
IdentifierChars: a..z A..Z _ 0..9 ?

// numeric constants begin characters
// note: characters shouldn't be delimited, except arrays
// array of chars could be defined as from_char..to_char
// number always starts with 0..9 except when NumConstBeg

```
// defines other
NumConstBegChars:      0..9

// numeric constants characters
// note: characters shouldn't be delimited, except arrays
// array of chars could be defined as from_char..to_char
// number always starts with 0..9 except when NumConstBeg
// defines other
NumConstChars:          0..9 .abcdefhABCDEFH

// escape character
EscapeChar:

// keyword table
// note: delimited with spaces, lines could be wrapped
// you may divide keywords into tree groups which can be
// highlighted differently

KeyWords1:                  aaa aad aam adc add and arpl bound bsf
                          bsr bswap bt btc btr bts call cbw cdq
                          more...

KeyWords2:

KeyWords3:

// string delimiter:
// StringBegChar - string begin char
// StringEndChar - string end char
// MultilineStrings - enables multiline strings,
// as perl has it
StringBegChar:          "
StringEndChar:          "
MultilineStrings:      0

// use preprocessor:
// 0 - no
// 1 - yes
// note: if yes, '#' and statements after it will be
// highlighted with Preprocessor defined colors
UsePreprocessor:       0

// highlight line:
// 0 - no
// 1 - yes
// note: if yes, current line will be highlighted
CurrLineHighlighted: 0

// colors
// note:                  first value is foreground, second is
//                         background color
//                           and third (optional) is font attribute:
//                           B - bold
//                           I - italic
//                           U - underline
//                           S - strike out
//                                attributes can be combined: eg. B or BI
//                             as value, it could be used any standard
//                             windows color:
//                                clBlack, clMaroon, clGreen, clOlive,
//                                clNavy, clPurple, clTeal, clGray,
//                                clSilver, clRed, clLime, clYellow,
//                                clBlue, clFuchsia, clAqua, clLtGray,
//                                clDkGray, clWhite, clScrollBar,
//                                clBackground, clActiveCaption,
//                                clInactiveCaption, clMenu, clWindow,
//                                clWindowFrame, clMenuText, clWindowText,
//                                clCaptionText, clActiveBorder,
//                                clInactiveBorder, clAppWorkSpace,
//                                clHighlight, clHighlightText, clBtnFace,
//                                clBtnShadow, clGrayText, lBtnText,
//                                clInactiveCaptionText, clBtnHighlight,
//                                cl3DDkShadow, cl3DLight, clInfoText,
//                                clInfoBk
//                             as value, it could be used hex numeric
//                             constant too:
//                                $BBGGRR - BB: blue, GG: green, RR: red,
//                                               eg: $FF6A00

SpaceCol:                 clWindowText clWindow
Keyword1Col:              clNavy clWindow
Keyword2Col:              clPurple clWindow
Keyword3Col:              clBlue clWindow
IdentifierCol:          clWindowText clWindow
CommentCol:                clGray clWindow
NumberCol:                clRed clWindow
StringCol:              clMaroon clWindow
SymbolCol:                clGray clWindow
PreprocessorCol:          clBlue clWindow
SelectionCol:            clWhite clNavy
// If CurrLineHighlighted: 1
CurrentLineCol:          clBlack clYellow
```

Code templates

Code template is a set of shortcuts with associated code. It is used for most frequently used code structures. Templates are stored in directory C:\Program Files\ConTEXT\Template\.

Format:

```
[user_name | description]
code
```

Note: Pipe char (|) in code defines where cursor will be positioned after inserting code from template.

Example

```
[ifb | if statement]
if | then
begin

end;
```

For edit code template, see: Options menu.

See ObjectPascal template for complete example.

If you have your own templates, send it to me to include it in next ConTEXT distributions.

Shortcuts

Cursor movement

Shortcut	Action
Ctrl+Left	Moves cursor left one word
Ctrl+Right	Moves cursor right one word
Ctrl+PageUp	Move cursor to the top of page
Ctrl+PageDown	Move cursor to the bottom of page
Ctrl+Home	Move cursor to the beginning of file
Ctrl+End	Move cursor to the end of file
Ctrl+Up	Scroll up one line leaving cursor position unchanged
Ctrl+Down	Scroll down one line leaving cursor position unchanged
Ctrl+G	Jumps to line
Ctrl+Shift+0..9	Set/clear bookmark
Ctrl+0..9	Go to bookmark

Editing

Shortcut	Action
Ctrl+A	Selects entire contents of editor
Ctrl+T	Delete from cursor to end of word
Ctrl+Backspace	Delete from cursor to start of word
Ctrl+Shift+Y	Delete from cursor to end of line
Ctrl+Y	Delete current line
Ctrl+Z	Undo
Ctrl+Shift+Z	Redo
Ctrl+C	Copy selection to clipboard
Ctrl+X	Cut selection to clipboard
Ctrl+V	Paste clipboard to current position
Ctrl+L	Toggle normal/column selection mode Hold down the ALT key while making your selection with the mouse for column select text
Ctrl+M	Go to matching brace
Ctrl+Shift+M	Select text between braces

File operations

Shortcut	Action
Ctrl+N	Create new file
Ctrl+O	Open file
Ctrl+S	Save file
F12	Save file as...
Ctrl+F4	Close file

Formatting

Shortcut	Action

Insert	Toggle insert/overwrite mode
F5	Toggle text case
Ctrl+F5	Toggle to lower case
Alt+F5	Toggle to upper case
Ctrl+Shift+I	Indent selection
Ctrl+Shift+U	Unindent selection
Ctrl+B	Reformat paragraph

Find/replace

Shortcut	**Action**
Ctrl+F	Find text
F3	Find next occurrence
Shift+F3	Find previous occurrence
Ctrl+R	Replace text

Macros

Shortcut	**Action**
Ctrl+F8	Record macro
F8	Select and play macro

Misc

Shortcut	**Action**
F9..F12	User defined commands
Ctrl+F12	Shell execute file with associated application
Alt+1..0	Jump to opened file
Ctrl+J	Insert code from template
Ctrl+Shift+C	Comment/Uncomment block or line of code

Notepad replacement

Since many applications has hard coded call to Windows Notepad.exe, the only way to start ConTEXT instead original Windows Notepad is to cheat those applications by replacing original Windows Notepad with other fake-Notepad which will call ConTEXT every time Notepad is started. To achieve this, follow next steps:

For Windows 95/98, NT

1. Backup your original notepad.exe found in C:\Windows\ (Windows 95/98) or in C:\WinNT\System32 (Windows NT) by renaming it to e.g. Notepad.bak
2. Copy Notepad.exe found in directory where ConTEXT is installed (e.g. C:\Program Files\ConTEXT\) to C:\Windows or C:\WinNT\System32\

For Windows 2000

1. Rename C:\WinNT\System32\DLLCache\Notepad.exe to Notepad.bak. If you don't see C:\WinNT\System32\DLLCache folder in explorer, start MS-DOS prompt and type:

   ```
   cd c:\WinNT\system32\dllcache\
   ren notepad.exe notepad.bak
   ```

2. Replace C:\WinNT\Notepad.exe and C:\WinNT\System32\Notepad.exe with Notepad.exe found in directory where ConTEXT is installed.
3. Windows will tell you that a system file has been replaced and ask you for the CD (since it can't refresh it automatically from dllcache).
   Just cancel the dialog and then press YES to let it know you want to keep the new file.

For Internet Explorer

1. Start IE and set ConTEXT as default HTML editor (Tools / Internet options / Programs / HTML editor).
2. After this, in file menu of IE you'll see command "Edit with ConTEXT". use this to view HTML files inside IE.

Language translations

ConTEXT supports multilanguage. All language files are stored in "language\ subdirectory" in directory where ConTEXT is installed.

If translation to your language doesn't exists and you want to contribute translation, please be free to translate language\Custom.lng file and e-mail it to foetus@aphex.fpz.hr and it will be included in next ConTEXT distribution.

Also, I'd be glad if you send me any corrections to existing translations.

To helper the translation and its update, download Translator.

Frequently Asked Questions

See FAQ.html file located in your ConTEXT directory.

Environment options - General

Backup file on save
If checked, file backup will be created when file is saved.

Use current directory
If selected, backup file will be created where original file is located.

Backup directory
If selected, backup file will be created in one, centralized directory. Default is Backup\.

Replace original extension with .BAK
If checked, original extension will be replaced with .BAK. Otherwise, ConTEXT will append .BAK to the file name

Remember editing positions
If checked, last editing position and used highlighter for each saved file will be restored next time file is open.

Remember last directory used
If checked, directory last used when opening or saving files will be default directory.

Remember Find/Replace dialog options
If checked, last used Find/Replace dialog parameters will be restored next time ConTEXT is started.

Detect file changes by other application
If checked, ConTEXT will check if editing file is changed by some other application or not. If changed, confirmation dialog will be shown. ConTEXT will check if file is changed when it's gaining focus from some other application. If many files are open, this option can slow-down ConTEXT a bit.

Automatically update changed files
If checked, editing file will be automatically reopen if changed by some other application without any confirmation dialog

Show find/replace information dialog
If checked, dialog will be shown if no more matches found.

Show user exec completion dialog
If checked, dialog will be show when user command execution is finished.

Minimize ConTEXT if no files open
If checked. ConTEXT will be automatically minimized if no files open.

Minimize to system tray
If checked, ConTEXT will be minimized to system tray.

Create new file when ConTEXT is open without file name
If checked and if no command line parameters are given, ConTEXT will create new file in editor when started.

Allow multiple instances
If checked, more than one instance of ConTEXT is allowed. If not, all files will be open in currently running instance.

Visible file tab icons

If checked, icons of associated applications will be shown in file tabs.

Multiline file tabs
If checked, multiline file tabs mode will be used.

Language
Select language from the list of supported languages. After changing language, restart of ConTEXT is required.

Environment options - Editor

Auto indent
If checked, positions the cursor under the first nonblank character of the preceding nonblank line when you press Enter

Drag-drop editing
If checked, moving and copying blocks of text by dragging with mouse are possible. If Ctrl key is pressed during dragging, text block will be duplicated. Otherwise, it will be moved.

Allow cursor after end of line
If checked, it's possible to position the cursor beyond the end of line.

Enhance HomeKey positioning
If checked, Home key has functionality as in MS Visual Studio. When key is pressed for the first time, cursor is positioned at the first non-blank character. When pressed second time, it jumps to the beginning of the line.

Find text at cursor
Places the text at the cursor into the "Find what" field in the Find/Replace dialog box when you choose Edit/Find.

Smart tabs
If checked, tabs to the first non-blank character in the preceding line.

Line numbers
If checked, line numbers are shown in the gutter line.

Undo after save
If checked,   allows you to retrieve changes after a save.

Hide mouse cursor when typing
If checked, mouse pointer will be hidden during typing. Move mouse to show pointer back.

Trim trailing spaces
If checked, all blank characters at the end of lines will be deleted.

Tab width
Set the character columns width that the cursor will move to each time you press Tab. This value is ignored when editing file if **Smart tabs** option is checked. If open file contains hard tab characters in it, they will be expanded to the value of tab width.

Block indent
Specify the number of spaces to indent a marked block.

Extra line spacing
Specify the number of pixels that will be added between lines of text in editor.

C/Java block indent
Specify the number of spaces to automatically indent/outdent a { } block used in C and Java code.

Insert caret
Specify the shape of cursor when   insert mode is active.

Overwrite caret

Specify the shape of cursor when overwrite mode is active.

Tabs save mode
ConTEXT doesn't have internally native support for real hard tabs, so tabs will be converted to spaces when opening file and spaces will be converted back to tabs when saving file. Behavior of this situation is controller by **Tabs save mode** option. If it's automatic, ConTEXT detects appearance of the tabs when opening files and when saving it, it uses tabs again wherever possible. You may also force tabs and spaces using this option.

Right margin
Specify the position of the column in which the text will be automatically wrapped to the next line when word-wrap mode is active. This value is used for Reformat paragraph command.

Gutter visible
Is checked, gutter line on the left side of editor window is visible.

Gutter width
Specify the width of the gutter line.

Default settings for new document:

Highlighter
Specify which highlighter will be used as default when new document is created.

File format
Specify which file format will be used as default when new document is created.

Environment options - Colors

Highlighter

List of highlighters available to ConTEXT.

Items list

List of syntax coloring items available for selected highlighter.

Foreground

Select foreground color for selected syntax item. If *Default color* selected, system clWindowText color will be used.

Background

Select background color for selected syntax item. If *Default color* selected, system clWindow color will be used.

Font attributes

Select text attributes for selected syntax item. If text attributes used with certain fonts (for example, Fixedsys), overall vertical space between characters will be increased.

Highlight current line

If checked, current line where cursor is positioned will be highlighted.

Override foreground text colors

If checked, current line foreground colors will be overriden. This option is active only if *Highlight current line* option checked. When this option is checked, ConTEXT needs restart!

Environment options - Execute Keys

File extension tree

This tree represents function key assignment for each file type. It is possible to assign four function key commands (F9-F12). To add new execute key command file type, press Add button. In dialog, enter list of file extensions for which this command assignment will be used.

Execute

Enter command that will be executed. To quickly browse to the executable file, press "..." browse button. This box can contain path and name to the executable file. If other file type is specified, ConTEXT will try to execute associated application. Don't specify any of the command line parameters in this box!

Start in

Specify directory that will be used as default when executing command. If omitted, default directory will be directory where is editing file.

Parameters

Place any command line parameters here.

Window

Select window type in which command will be used. If executing application is console or DOS application and Capture console output option is checked, this parameter will be ignored.

Hint

Enter some text that will describe this command. This text will appear as Tools menu command caption and as hint on corresponding toolbar button.

Save

Select save mode that will be used before command is executed.

Use short DOS names

If checked, all paths and file names will be converted to the old DOS 8.3 file name format. Check this option if you're executing DOS application that can't handle long file names.

Capture console output

If checked, all output of the executed command will be redirected to the small window in the bottom of ConTEXT instead in separate console window. You should be aware of following notes:

1. If executed command enters infinite loop and you're using Windows NT or Windows 2000, Terminate button will be shown in console output window and executing command could be terminated.

2. If you use Windows 9x/Me, Terminate function wouldn't be available and if application enters infinite loop, you should use task manager or some third party utility to terminate the process.

3. If executed application needs user input, this will be NO indicated and application will probably look as locked.

Compiler output parser rule

See Compiler output parser.

Optional generic runtime parameters

Runtime parameters can be used in any of above fields:

%n	file name with path
%f	file name only, without path
%F	file name only, without path and extension
%p	file path only
%e	file extension only
%s	value specified in "Start in" box
%P	file specific parameters - see topic below for more info on this parameter
%l	current line number
%c	current column number
%w	word under cursor
%%	percent sign
%?	executes parameters input dialog where you can enter additional parameters

File specific parameters

Parameter %P is file specific parameter and it can be used for every edited file separately. In the **first** line of code following comment should be added:

```
// %PARAMETERS = "my additonal parameters"
```

Of course, replace '//' comment sequence with comment chars of language you use. You can use escape (\) character to enter quotes (\") or backslash (\\).

For example, if Parameters in Environment options dialog is defined as "%n %P blabla %e" and first line of code contains definition such as %PARAMETERS = "-first /second -\"third\"", your compiler (or whatever) will be started with following parameters:

```
mycompiler.exe c:\test\test.ext -first /second -"third" ext
```

Examples

Make file backup

Execute:	C:\WinNT\system32\cmd.exe
Parameters:	/c copy %n %n.bak

This command will copy current editing file to the new file and append extension .bak to it.

Find current word in all files in current directory

Execute:	C:\WinNT\system32\find.exe
Parameters:	/i /n "%w" %p*.*
Capture console output:	ON

This command will search through all files in current directory, locate all matches of word under the cursor and show it in captured console output window.

Compile pascal file using TurboPascal compiler:

Execute:	C:\Program Files\util\TurboPascal\tpc.exe
Parameters:	%n
Save: Current file before execution	
Use short names:	ON
Capture console output:	ON
Compiler output parser rule:	%n(%l)

This command will invoke TurboPascal compiler, compile current editing file, capture compiler output and parse it using %n(%l) rule.

Compile assembler file using TASM:

Execute:	C:\Program Files\util\compilers\tasm.exe
Parameters:	-Ic:\progra~1\compil~1 -Lc:\progra~1\compil~1 %n
Save:	Current file before execution
Use short names:	ON
Capture console output:	ON
Compiler output parser rule:	%n(%l)

This command will invoke TurboAssembler compiler, compile current editing file, capture compiler output and parse it using %n(%l) rule.

Environment options - Associations

Add

Use this command to associate file type to ConTEXT.

Remove

Use this command to remove file type association.

Environment options - Miscellaneous

Output console font
Define captured console output font.

Default SQL dialect
Select default SQL dialect that will be used as syntax highlighter scheme when SQL highlighter is active.

Help files
You can assign help file for each file type. Help file can be invoked by pressing F1 key.

Regular expressions

ConTEXT is using TRegExp Delphi library written by Andrey V. Sorokin <anso@mail.ru>. It can be downloaded from http://anso.da.ru (http://anso.virtualave.net).

Introduction

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special metacharacters allow You to specify, for instance, that a particular string You are looking for occurs at the beginning or end of a line, or contains n recurrences of a certain character.

Regular expressions look ugly for novices, but really they are very simple (well, usually simple ;)), handly and powerfull tool.

I strongly recommend You to play with regular expressions using TestRExp.dpr - it'll help You to uderstand main conceptions. Moreover, there are many predefined examples with comments included into TestRExp.

Let's start our learning trip!

Simple matches

Any single character matches itself, unless it is a metacharacter with a special meaning described below.

A series of characters matches that series of characters in the target string, so the pattern "bluh" would match "bluh'' in the target string. Quite simple, eh ?

You can cause characters that normally function as metacharacters or escape sequences to be interpreted literally by 'escaping' them by preceding them with a backslash "\", for instance: metacharacter "^" match beginning of string, but "\^" match character "^", "\\" match "\" and so on.

Examples:

```
  foobar              matchs string 'foobar'
  \^FooBarPtr         matchs '^FooBarPtr'
```

Escape sequences

Characters may be specified using a escape sequences syntax much like that used in C and Perl: "\n'' matches a newline, "\t'' a tab, etc. More generally, \xnn, where nn is a string of hexadecimal digits, matches the character whose ASCII value is nn. If You need wide (Unicode) character code, You can use '\x{nnnn}', where 'nnnn' - one or more hexadecimal digits.

```
  \xnn       char with hex code nn
  \x{nnnn} char with hex code nnnn (one byte for plain text and two bytes for Unicode)
  \t         tab (HT/TAB), same as \x09
  \n         newline (NL), same as \x0a
  \r         car.return (CR), same as \x0d
  \f         form feed (FF), same as \x0c
  \a         alarm (bell) (BEL), same as \x07
  \e         escape (ESC), same as \x1b
```

Examples:

```
  foo\x20bar     matchs 'foo bar' (note space in the middle)
  \tfoobar       matchs 'foobar' predefined by tab
```

Character classes

You can specify a character class, by enclosing a list of characters in [], which will match any one character from the list.

If the first character after the "[" is "^", the class matches any character not in the list.

Examples:

foob[aeiou]r	finds strings 'foobar', 'foober' etc. but not 'foobbr', 'foobcr' etc.
foob[^aeiou]r	find strings 'foobbr', 'foobcr' etc. but not 'foobar', 'foober' etc.

Within a list, the "-" character is used to specify a range, so that a-z represents all characters between "a" and "z", inclusive.

If You want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash. If You want ']' you may place it at the start of list or escape it with a backslash.

Examples:

[-az]	matchs 'a', 'z' and '-'
[az-]	matchs 'a', 'z' and '-'
[a\-z]	matchs 'a', 'z' and '-'
[a-z]	matchs all twenty six small characters from 'a' to 'z'
[\n-\x0D]	matchs any of #10,#11,#12,#13.
[\d-t]	matchs any digit, '-' or 't'.
[]-a]	matchs any char from ']'..'a'.

Metacharacters

Metacharacters are special characters which are the essence of Regular Expressions. There are different types of metacharacters, described below.

Metacharacters - line separators

^	start of line
$	end of line
\A	start of text
\Z	end of text
.	any character in line

Examples:

^foobar	matchs string 'foobar' only if it's at the beginning of line
foobar$	matchs string 'foobar' only if it's at the end of line
^foobar$	matchs string 'foobar' only if it's the only string in line
foob.r	matchs strings like 'foobar', 'foobbr', 'foob1r' and so on

The "^" metacharacter by default is only guaranteed to match at the beginning of the input string/text, the "$" metacharacter only at the end. Embedded line separators will not be matched by "^" or "$".
You may, however, wish to treat a string as a multi-line buffer, such that the "^" will match after any line separator within the string, and "$" will match before any line separator. You can do this by switching On the modifier /m.

The \A and \Z are just like "^" and "$", except that they won't match multiple times when the modifier /m is used, while "^" and "$" will match at every internal line separator.

The "." metacharacter by default matches any character, but if You switch Off the modifier /s, then '.' won't match embedded line separators.

TRegExpr works with line separators as recommended at www.unicode.org (http://www.unicode.org/unicode/reports/tr18/):

"^" is at the beginning of a input string, and, if modifier /m is On, also immediately following any occurrence of \x0D\x0A or \x0A or \x0D (if You are using Unicode version of TRegExpr, then also \x2028 or \x2029 or \x0B or \x0C or \x85). Note that there is no empty line within the sequence \x0D\x0A.

"$" is at the end of a input string, and, if modifier /m is On, also immediately preceding any occurrence of \x0D\x0A or \x0A or \x0D (if You are using Unicode version of TRegExpr, then also \x2028 or \x2029 or \x0B or \x0C or \x85). Note that there is no empty line within the sequence \x0D\x0A.

"." matchs any character, but if You switch Off modifier /s then "." doesn't match \x0D\x0A and \x0A and \x0D (if You are using Unicode version
of TRegExpr, then also \x2028 and \x2029 and \x0B and \x0C and \x85).

Note that "^.*$" (an empty line pattern) doesnot match the empty string within the sequence \x0D\x0A, but matchs the empty string within the sequence \x0A\x0D.

Multiline processing can be easely tuned for Your own purpose with help of TRegExpr properties LineSeparators and LinePairedSeparator, You can use only Unix style separators \n or only DOS/Windows style \r\n or mix them together (as described above and used by default) or define Your own line separators!

Metacharacters - predefined classes

\w	an alphanumeric character (including "_")
\W	a nonalphanumeric
\d	a numeric character
\D	a non-numeric
\s	any space (same as [\t\n\r\f])
\S	a non space

You may use \w, \d and \s within custom character classes.

Examples:

foob\dr	matchs strings like 'foob1r', ''foob6r' and so on but not 'foobar', 'foobbr' and so on
foob[\w\s]r	matchs strings like 'foobar', 'foob r', 'foobbr' and so on but not 'foob1r', 'foob=r' and so on

TRegExpr uses properties SpaceChars and WordChars to define character classes \w, \W, \s, \S, so You can easely redefine it.

Metacharacters - word boundaries

\b	Match a word boundary
\B	Match a non-(word boundary)

A word boundary (\b) is a spot between two characters that has a \w on one side of it and a \W on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a \W.

Metacharacters - iterators

Any item of a regular expression may be followed by another type of metacharacters - iterators. Using this metacharacters You can specify number of occurences of previous character, metacharacter or subexpression.

*	zero or more ("greedy"), similar to {0,}
+	one or more ("greedy"), similar to {1,}
?	zero or one ("greedy"), similar to {0,1}
{n}	exactly n times ("greedy")
{n,}	at least n times ("greedy")
{n,m}	at least n but not more than m times ("greedy")
*?	zero or more ("non-greedy"), similar to {0,}?
+?	one or more ("non-greedy"), similar to {1,}?
??	zero or one ("non-greedy"), similar to {0,1}?
{n}?	exactly n times ("non-greedy")
{n,}?	at least n times ("non-greedy")
{n,m}?	at least n but not more than m times ("non-greedy")

So, digits in curly brackets of the form {n,m}, specify the minimum number of times to match the item n and the maximum m. The form {n} is equivalent to {n,n} and matches exactly n times. The form {n,} matches n or more times. There is no limit to the size of n or m, but large numbers will chew up more memory and slow down r.e. execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

Examples:

foob.*r	matchs strings like 'foobar', 'foobalkjdflkj9r' and 'foobr'
foob.+r	matchs strings like 'foobar', 'foobalkjdflkj9r' but not 'foobr'
foob.?r	matchs strings like 'foobar', 'foobbr' and 'foobr' but not 'foobalkj9r'
fooba{2}r	matchs the string 'foobaar'
fooba{2,}r	matchs strings like 'foobaar', 'foobaaar', 'foobaaaar' etc.
fooba{2,3}r	matchs strings like 'foobaar', or 'foobaaar' but not 'foobaaaar'

A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible. For example, 'b+' and 'b*' applied to string 'abbbbc' return 'bbbb', 'b+?' returns 'b', 'b*?' returns empty string, 'b{2,3}?' returns 'bb', 'b{2,3}' returns 'bbb'.

You can switch all iterators into "non-greedy" mode (see the modifier /g).

Metacharacters - alternatives

You can specify a series of alternatives for a pattern using "|" to separate them, so that fee|fie|foe will match any of "fee", "fie", or "foe" in the target string (as would f(e|i|o)e). The first alternative includes everything from the last pattern delimiter ("(", "[", or the beginning of the pattern) up to the first "|", and the last alternative contains everything from the last "|" to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching foo|foot against "barefoot", only the "foo" part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.)

Also remember that "|" is interpreted as a literal within square brackets, so if You write [fee|fie|foe] You're really only matching [feio|].

Examples:
```
  foo(bar|foo)   matchs strings 'foobar' or 'foofoo'.
```

Metacharacters - subexpressions

The bracketing construct (...) may also be used for define r.e. subexpressions (after parsing You can find subexpression positions, lengths and actual values in MatchPos, MatchLen and Match properties of TRegExpr, and substitute it in template strings by TRegExpr.Substitute).

Subexpressions are numbered based on the left to right order of their opening parenthesis.
First subexpression has number '1' (whole r.e. match has number '0' - You can substitute it in TRegExpr.Substitute as '$0' or '$&').

Examples:
```
  (foobar){8,10}   matchs strings which contain 8, 9 or 10 instances of the 'foobar'
  foob([0-9]|a+)r matchs 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.
```

Metacharacters - backreferences

Metacharacters \1 through \9 are interpreted as backreferences. \<n> matches previously matched subexpression #<n>.

Examples:
```
  (.)\1+            matchs 'aaaa' and 'cc'.
  (.+)\1+           also match 'abab' and '123123'
  (['"]?)(\d+)\1 matchs '"13" (in double quotes), or '4' (in single quotes) or 77 (without quotes) etc
```

Modifiers

Modifiers are for changing behaviour of TRegExpr.

There are many ways to set up modifiers.
Any of these modifiers may be embedded within the regular expression itself using the (?...) construct.
Also, You can assign to appropriate TRegExpr properties (ModifierX for example to change /x, or ModifierStr to change all modifiers together). The default values for new instances of TRegExpr object defined in global variables, for example global variable RegExprModifierX defines value of new TRegExpr instance ModifierX property.

i

Do case-insensitive pattern matching (using installed in you system locale settings), see also InvertCase.

m

Treat string as multiple lines. That is, change "^" and "$" from matching at only the very start or end of the string to the start or end of any line anywhere within the string, see also Line separators.

s

Treat string as single line. That is, change "." to match any character whatsoever, even a line separators (see also Line separators), which it normally would not match.

g

Non standard modifier. Switching it Off You'll switch all following operators into non-greedy mode (by default this modifier is On). So, if modifier /g is Off then '+' works as '+?', '*' as '*?' and so on

x

Extend your pattern's legibility by permitting whitespace and comments (see explanation below).

r

Non-standard modifier. If is set then range à-ÿ additional include russian letter '¸', À-ß   additional include '¨', and à-ß include all russian symbols.
Sorry for foreign users, but it's set by default. If you want switch if off by default - set false to global variable RegExprModifierR.

The modifier /x itself needs a little more explanation. It tells the TRegExpr to ignore whitespace that is neither backslashed nor within a character class. You can use this to break up your regular expression into (slightly) more readable parts. The # character is also treated as a metacharacter introducing a comment, for example:

```
(
(abc) # comment 1
   |     # You can use spaces to format r.e. - TRegExpr ignores it
(efg) # comment 2
)
```

This also means that if you want real whitespace or # characters in the pattern (outside a character class, where they are unaffected by /x), that you'll either have to escape them or encode them using octal or hex escapes. Taken together, these features go a long way towards making regular expressions text more readable.

Perl extensions

(?imsxr-imsxr)
You may use it into r.e. for modifying modifiers by the fly. If this construction inlined into subexpression, then it effects only into this subexpression

Examples:

```
   (?i)Saint-Petersburg          matchs 'Saint-petersburg' and 'Saint-Petersburg'
   (?i)Saint-(?-i)Petersburg   matchs 'Saint-Petersburg' but not 'Saint-petersburg'
   (?i)(Saint-)?Petersburg      matchs 'Saint-petersburg' and 'saint-petersburg'

   ((?i)Saint-)?Petersburg      matchs 'saint-Petersburg', but not 'saint-petersburg'
```

(?#text)
A comment, the text is ignored. Note that TRegExpr closes the comment as soon as it sees a ")", so there is no way to put a literal ")" in the comment.