fb-doc

0.3.9

Generated by Doxygen 1.8.4

Sat May 17 2014 19:05:48

Contents

1 fb-doc **1**

1.1 fb-doc licence (GPLv3): . . . 1

1.2 Acknowledgements . . . 2

2 Further Development **3**

3 Emitters in Detail **5**

3.1 C Source . . . 5

3.2 GtkDoc Templates . . . 5

3.3 Doxygen Templages . . . 6

3.4 List of Function Names . . . 6

3.5 Syntax Hightlighting . . . 6

4 Examples **9**

4.1 Command Line Interface . . . 9

4.2 Comments (in general) . . . 10

4.3 Variables Declaration . . . 12

4.4 Templates . . . 12

4.4.1 Gtk-doc . . . 13

4.4.2 Doxygen . . . 15

5 Extending fb-doc **19**

5.1 Internals . . . 19

5.2 Control Flow . . . 20

5.3 Example Plugin.bas . . . 20

6 Files and Folders **23**

6.1 ../doc/c_src . . . 23

6.2 fb-doc.lfn . . . 23

6.3 External Moduls (∗.so / ∗.dll) . . . 23

7 Compiling and Installing **25**

7.1 UNIX Installation . . . 25

7.2 DOS / windows Installation . . . 26

7.3 Geany IDE Installation . . . 26

8 Introduction **29**

8.1 fb-doc . . . 29
8.2 Documentation Context . . . 30
8.3 About this Text . . . 30

9 Options in Detail **31**

9.1 Mode Options . . . 31
9.1.1 Default Mode . . . 32
9.1.2 File Mode (-f) . . . 33
9.1.3 Geany Mode (-g) . . . 34
9.1.4 List Mode (-l) . . . 35
9.1.5 Syntax Mode (-s) . . . 35
9.1.6 Help (-h) . . . 36
9.1.7 Version (-v) . . . 36
9.2 Operational Options . . . 36
9.2.1 Asterix (-a) . . . 36
9.2.2 CStyle (-c) . . . 37
9.2.3 Emitter (-e) . . . 37
9.2.4 Outpath (-o) . . . 37
9.2.5 Recursiv (-r) . . . 38
9.2.6 Tree (-t) . . . 38
9.3 File Specifications . . . 39

10 Tables **41**

10.1 Overview . . . 41
10.2 Options . . . 41
10.3 Inbuild Emitters . . . 42
10.4 Input / Output . . . 43
10.5 File Specifications . . . 43
10.6 Intermediate Format . . . 44
10.7 Files . . . 44

11 How To Use **47**

11.1 Generating Templates . . . 47
11.2 Gtk-doc . . . 48
11.3 Doxygen Back-End . . . 48
11.4 C Headers . . . 50
11.5 Piping . . . 50

12 Class Index **51**

12.1 Class List . . . 51

13 File Index **53**

13.1 File List . . . 53

14 Class Documentation **55**

14.1 Doxyfile Class Reference . . . 55
14.1.1 Detailed Description . . . 56
14.1.2 Constructor & Destructor Documentation . . . 56
14.1.2.1 Doxyfile . . . 56
14.1.2.2 ~Doxyfile . . . 56
14.1.3 Member Function Documentation . . . 56
14.1.3.1 Tag . . . 56
14.1.4 Member Data Documentation . . . 57
14.1.4.1 Length . . . 57
14.1.4.2 Buffer . . . 57
14.1.4.3 Doxy . . . 57
14.1.4.4 Errr . . . 57
14.2 EmitterIF Class Reference . . . 57
14.2.1 Detailed Description . . . 59
14.2.2 Member Function Documentation . . . 59
14.2.2.1 Init_ . . . 59
14.2.2.2 Decl_ . . . 59
14.2.2.3 Func_ . . . 60
14.2.2.4 Enum_ . . . 61
14.2.2.5 Unio_ . . . 61
14.2.2.6 Clas_ . . . 61
14.2.2.7 Defi_ . . . 62
14.2.2.8 Incl_ . . . 62
14.2.2.9 Error_ . . . 63
14.2.2.10 Empty_ . . . 63
14.2.2.11 Exit_ . . . 64
14.2.2.12 CTOR_ . . . 64
14.2.2.13 DTOR_ . . . 64
14.2.3 Member Data Documentation . . . 64
14.2.3.1 Nam . . . 64
14.3 Highlighter Class Reference . . . 65
14.3.1 Detailed Description . . . 67
14.3.2 Member Enumeration Documentation . . . 67
14.3.2.1 WordTypes . . . 67
14.3.3 Constructor & Destructor Documentation . . . 68

14.3.3.1 Highlighter . . . 68
14.3.3.2 Highlighter . . . 68
14.3.4 Member Function Documentation . . . 69
14.3.4.1 doDoxy . . . 69
14.3.4.2 do_files . . . 70
14.3.4.3 prepare_tex . . . 71
14.3.4.4 prepare_xml . . . 71
14.3.4.5 prepare_html . . . 72
14.3.4.6 generate_all . . . 72
14.3.4.7 generate_code . . . 73
14.3.4.8 word_type . . . 74
14.3.4.9 eol . . . 74
14.3.4.10 prepare . . . 74
14.3.4.11 special_chars . . . 74
14.3.5 Member Data Documentation . . . 74
14.3.5.1 FbPath . . . 74
14.3.5.2 FbFiles . . . 75
14.3.5.3 InPath . . . 75
14.3.5.4 DoxyFiles . . . 75
14.3.5.5 HtmlPath . . . 75
14.3.5.6 HtmlSuff . . . 75
14.3.5.7 TexPath . . . 75
14.3.5.8 XmlPath . . . 75
14.3.5.9 LastLine . . . 75
14.3.5.10 Symbols . . . 75
14.3.5.11 Pars . . . 76
14.3.5.12 FBDOC_MARK . . . 76
14.3.5.13 KEYW_A . . . 76
14.3.5.14 KWTP_A . . . 76
14.3.5.15 KWFL_A . . . 76
14.3.5.16 PREP_A . . . 76
14.3.5.17 CMNT_A . . . 76
14.3.5.18 SPAN_E . . . 76
14.3.5.19 QUOT_A . . . 76
14.3.5.20 QUOT_E . . . 77
14.3.5.21 Ifnr . . . 77
14.3.5.22 LineNo . . . 77
14.3.5.23 "@1 . . . 77
14.4 Options Class Reference . . . 77
14.4.1 Detailed Description . . . 80

14.4.2 Member Enumeration Documentation . . . 80
14.4.2.1 RunModes . . . 80
14.4.2.2 EmitterTypes . . . 81
14.4.2.3 TypesStyle . . . 81
14.4.2.4 CaseModes . . . 82
14.4.3 Constructor & Destructor Documentation . . . 82
14.4.3.1 Options . . . 82
14.4.3.2 ~Options . . . 82
14.4.4 Member Function Documentation . . . 82
14.4.4.1 parseCLI . . . 82
14.4.4.2 parseOptpara . . . 83
14.4.4.3 chooseEmitter . . . 84
14.4.4.4 FileModi . . . 85
14.4.4.5 doFile . . . 86
14.4.4.6 checkDir . . . 87
14.4.4.7 scanFiles . . . 88
14.4.4.8 addPath . . . 89
14.4.5 Member Data Documentation . . . 89
14.4.5.1 RunMode . . . 89
14.4.5.2 Types . . . 89
14.4.5.3 Pars . . . 89
14.4.5.4 EmitTyp . . . 89
14.4.5.5 CaseMode . . . 90
14.4.5.6 EmitIF . . . 90
14.4.5.7 DllEmitter . . . 90
14.4.5.8 DirUp . . . 90
14.4.5.9 InFiles . . . 90
14.4.5.10 FileName . . . 90
14.4.5.11 StartPath . . . 90
14.4.5.12 FileIncl . . . 90
14.4.5.13 OutPath . . . 90
14.4.5.14 Errr . . . 91
14.4.5.15 Asterix . . . 91
14.4.5.16 AllCallees . . . 91
14.4.5.17 InRecursiv . . . 91
14.4.5.18 InTree . . . 91
14.4.5.19 Level . . . 91
14.4.5.20 Efnr . . . 91
14.4.5.21 Ocha . . . 91
14.4.5.22 JoComm . . . 91

14.4.5.23 AdLine . . . 92
14.4.5.24 CreateFunction . . . 92
14.4.5.25 CreateVariable . . . 92
14.5 Parser Class Reference . . . 92
14.5.1 Detailed Description . . . 97
14.5.2 Member Enumeration Documentation . . . 97
14.5.2.1 ParserTokens . . . 97
14.5.2.2 EoS_Modi . . . 99
14.5.3 Constructor & Destructor Documentation . . . 99
14.5.3.1 Parser . . . 99
14.5.4 Member Function Documentation . . . 100
14.5.4.1 File_ . . . 100
14.5.4.2 StdIn . . . 101
14.5.4.3 Include . . . 102
14.5.4.4 writeOut . . . 103
14.5.4.5 CurTok . . . 104
14.5.4.6 BitIni . . . 104
14.5.4.7 SubStr . . . 105
14.5.4.8 SubStr . . . 107
14.5.4.9 parseListNam . . . 108
14.5.4.10 parseListNamTyp . . . 108
14.5.4.11 parseListPara . . . 109
14.5.4.12 parseBlockEnum . . . 110
14.5.4.13 parseBlockTyUn . . . 111
14.5.4.14 pre_parse . . . 112
14.5.4.15 USubStr . . . 113
14.5.4.16 getToken . . . 114
14.5.4.17 tokenize . . . 115
14.5.4.18 Errr . . . 115
14.5.4.19 demuxNam . . . 116
14.5.4.20 demuxTyp . . . 117
14.5.4.21 demuxDecl . . . 118
14.5.4.22 skipOverColon . . . 118
14.5.4.23 skipOverBrclo . . . 119
14.5.4.24 skipOverComma . . . 119
14.5.4.25 VAR_ . . . 120
14.5.4.26 TYPE_ . . . 121
14.5.4.27 ENUM_ . . . 122
14.5.4.28 UNION_ . . . 122
14.5.4.29 FUNCTION_ . . . 123

14.5.4.30 DECLARE_ . . . 124
14.5.4.31 INCLUDE_ . . . 125
14.5.4.32 DEFINE_ . . . 126
14.5.4.33 MACRO_ . . . 126
14.5.5 Member Data Documentation . . . 127
14.5.5.1 InPath . . . 127
14.5.5.2 Buf . . . 127
14.5.5.3 ErrMsg . . . 127
14.5.5.4 Tok . . . 127
14.5.5.5 BlockNam . . . 128
14.5.5.6 StaTok . . . 128
14.5.5.7 NamTok . . . 128
14.5.5.8 DimTok . . . 128
14.5.5.9 IniTok . . . 128
14.5.5.10 BitTok . . . 128
14.5.5.11 TypTok . . . 128
14.5.5.12 ShaTok . . . 128
14.5.5.13 PtrTok . . . 128
14.5.5.14 Co1Tok . . . 129
14.5.5.15 Co2Tok . . . 129
14.5.5.16 FunTok . . . 129
14.5.5.17 CalTok . . . 129
14.5.5.18 AliTok . . . 129
14.5.5.19 As_Tok . . . 129
14.5.5.20 ParTok . . . 129
14.5.5.21 By_Tok . . . 129
14.5.5.22 DivTok . . . 129
14.5.5.23 Tk1 . . . 130
14.5.5.24 EndTok . . . 130
14.5.5.25 UserTok . . . 130
14.5.5.26 Po . . . 130
14.5.5.27 Fin . . . 130
14.5.5.28 PtrCount . . . 130
14.5.5.29 SrcBgn . . . 130
14.5.5.30 LineNo . . . 130
14.5.5.31 LevelCount . . . 130
14.5.5.32 InTree . . . 131
14.5.5.33 ListCount . . . 131
14.5.5.34 ToLast . . . 131
14.5.5.35 Tk . . . 131

14.5.5.36 A . . . 131
14.5.5.37 L . . . 131
14.5.5.38 Emit . . . 131
14.6 RepData Class Reference . . . 131
14.6.1 Detailed Description . . . 132
14.6.2 Member Function Documentation . . . 132
14.6.2.1 add . . . 132
14.6.2.2 rep . . . 133
14.6.3 Member Data Documentation . . . 134
14.6.3.1 O . . . 134
14.6.3.2 I . . . 134

15 File Documentation **135**

15.1 src/Development.md File Reference . . . 135
15.2 src/Development.md . . . 135
15.3 src/Emitters.md File Reference . . . 135
15.4 src/Emitters.md . . . 135
15.5 src/Examples.md File Reference . . . 137
15.6 src/Examples.md . . . 137
15.7 src/Extend.md File Reference . . . 144
15.8 src/Extend.md . . . 144
15.9 src/fb-doc.bas File Reference . . . 146
15.9.1 Detailed Description . . . 147
15.9.2 Function Documentation . . . 147
15.9.2.1 main . . . 147
15.9.3 Variable Documentation . . . 147
15.9.3.1 PROG_NAME . . . 147
15.9.3.2 PROG_VERS . . . 147
15.10 fb-doc.bas . . . 148
15.11 src/fb-doc.bi File Reference . . . 149
15.11.1 Detailed Description . . . 150
15.11.2 Macro Definition Documentation . . . 150
15.11.2.1 MSG_HELP . . . 150
15.11.2.2 MSG_WELCOME . . . 151
15.11.2.3 MSG_VERSION . . . 151
15.11.2.4 FBDOC_TARGET . . . 151
15.11.2.5 FBDOC_BINARY . . . 151
15.11.2.6 FBDOC_BINARY . . . 152
15.11.3 Variable Documentation . . . 152
15.11.3.1 COMM_END . . . 152

15.11.3.2 FIXME . . . 152
15.12 fb-doc.bi . . . 152
15.13 src/fb-doc_doxyfile.bas File Reference . . . 153
15.13.1 Detailed Description . . . 153
15.14 fb-doc_doxyfile.bas . . . 153
15.15 src/fb-doc_emit_callees.bas File Reference . . . 155
15.15.1 Detailed Description . . . 155
15.15.2 Function Documentation . . . 156
15.15.2.1 writeLFN . . . 156
15.15.2.2 callees_decl_ . . . 156
15.15.2.3 callees_class_ . . . 157
15.15.2.4 callees_func_ . . . 157
15.15.2.5 callees_include . . . 158
15.15.3 Variable Documentation . . . 159
15.15.3.1 CALLEE_TR . . . 159
15.16 fb-doc_emit_callees.bas . . . 159
15.17 src/fb-doc_emit_doxy.bas File Reference . . . 160
15.17.1 Detailed Description . . . 161
15.17.2 Function Documentation . . . 162
15.17.2.1 doxy_entryListPara . . . 162
15.17.2.2 doxy_func_ . . . 162
15.17.2.3 doxy_decl_ . . . 163
15.17.2.4 doxy_defi_ . . . 164
15.17.2.5 doxy_emitBlockNames . . . 165
15.17.2.6 doxy_Block . . . 166
15.17.2.7 doxy_empty . . . 167
15.17.3 Variable Documentation . . . 167
15.17.3.1 DOXY_START . . . 167
15.17.3.2 DOXY_END . . . 167
15.18 fb-doc_emit_doxy.bas . . . 168
15.19 src/fb-doc_emit_gtk.bas File Reference . . . 171
15.19.1 Detailed Description . . . 172
15.19.2 Function Documentation . . . 172
15.19.2.1 gtk_emit_Name . . . 172
15.19.2.2 gtk_defi_ . . . 173
15.19.2.3 gtk_decl_ . . . 173
15.19.2.4 gtk_func_ . . . 174
15.19.2.5 gtk_emitBlockNames . . . 175
15.19.2.6 gtk_Block . . . 176
15.19.2.7 gtk_empty . . . 177

15.19.3 Variable Documentation . . . 178
15.19.3.1 SINCE . . . 178
15.19.3.2 GTK_START . . . 178
15.19.3.3 GTK_END . . . 178
15.20 fb-doc_emit_gtk.bas . . . 178
15.21 src/fb-doc_emit_syntax.bas File Reference . . . 181
15.21.1 Detailed Description . . . 182
15.21.2 Class Documentation . . . 182
15.21.2.1 union Highlighter.__unnamed__ . . . 182
15.21.2.2 class Highlighter.__unnamed__.__unnamed__ . . . 182
15.21.3 Macro Definition Documentation . . . 183
15.21.3.1 GET_WORD_TYPE . . . 183
15.21.4 Function Documentation . . . 183
15.21.4.1 html_specials . . . 183
15.21.4.2 tex_specials . . . 184
15.21.4.3 xml_specials . . . 184
15.21.4.4 html_eol . . . 184
15.21.4.5 tex_eol . . . 185
15.21.4.6 xml_eol . . . 186
15.21.4.7 synt_init . . . 186
15.21.4.8 synt_exit . . . 187
15.21.4.9 synt_incl . . . 187
15.21.4.10 synt_func_ . . . 188
15.22 fb-doc_emit_syntax.bas . . . 189
15.23 src/fb-doc_emitters.bas File Reference . . . 207
15.23.1 Detailed Description . . . 209
15.23.2 Function Documentation . . . 209
15.23.2.1 cEmitComments . . . 209
15.23.2.2 cppNam . . . 210
15.23.2.3 cNam . . . 211
15.23.2.4 cIni . . . 212
15.23.2.5 cArrDim . . . 212
15.23.2.6 cppCreateTypNam . . . 213
15.23.2.7 cCreateTypNam . . . 214
15.23.2.8 cppCreateFunction . . . 215
15.23.2.9 cppEntryListParameter . . . 216
15.23.2.10 cCreateFunction . . . 217
15.23.2.11 cEntryListParameter . . . 218
15.23.2.12 c_include . . . 219
15.23.2.13 c_defi_ . . . 220

15.23.2.14c_func_ . . . 220
15.23.2.15c_decl_ . . . 221
15.23.2.16cEntryBlockENUM . . . 222
15.23.2.17c_Block . . . 223
15.23.2.18cEntryBlockTypeUnion . . . 224
15.23.2.19c_error . . . 224
15.23.2.20c_Init . . . 225
15.23.2.21c_exit . . . 225
15.23.2.22c_CTOR . . . 226
15.23.2.23cEmitSource . . . 226
15.23.2.24geanyInit . . . 227
15.23.2.25geanyExit . . . 228
15.23.3 Variable Documentation . . . 228
15.23.3.1 LOFN . . . 228
15.24fb-doc_emitters.bas . . . 228
15.25src/fb-doc_emitters.bi File Reference . . . 238
15.25.1 Detailed Description . . . 240
15.25.2 Macro Definition Documentation . . . 240
15.25.2.1 WITH_NEW_EMITTER . . . 240
15.25.3 Typedef Documentation . . . 240
15.25.3.1 Parser_ . . . 240
15.25.3.2 EmitFunc . . . 240
15.25.4 Function Documentation . . . 240
15.25.4.1 null_emitter . . . 240
15.25.5 Variable Documentation . . . 240
15.25.5.1 Emitters . . . 241
15.26fb-doc_emitters.bi . . . 241
15.27src/fb-doc_options.bas File Reference . . . 242
15.27.1 Detailed Description . . . 243
15.28fb-doc_options.bas . . . 243
15.29src/fb-doc_options.bi File Reference . . . 249
15.29.1 Detailed Description . . . 250
15.29.2 Macro Definition Documentation . . . 251
15.29.2.1 ERROUT . . . 251
15.29.2.2 MSG_LINE . . . 251
15.29.2.3 MSG_END . . . 251
15.29.3 Variable Documentation . . . 251
15.29.3.1 CALLEES_FILE . . . 251
15.29.3.2 OPT . . . 251
15.30fb-doc_options.bi . . . 251

15.31 src/fb-doc_parser.bas File Reference . . . 253
15.31.1 Detailed Description . . . 255
15.31.2 Macro Definition Documentation . . . 255
15.31.2.1 SKIP . . . 255
15.31.2.2 SCAN_QUOTE . . . 255
15.31.2.3 SCAN_SL_COMM . . . 255
15.31.2.4 SCAN_ML_COMM . . . 255
15.31.2.5 SCAN_WORD . . . 256
15.31.2.6 SETOK . . . 256
15.31.2.7 EXIT_STOP . . . 256
15.32 fb-doc_parser.bas . . . 256
15.33 src/fb-doc_parser.bi File Reference . . . 272
15.33.1 Detailed Description . . . 273
15.33.2 Macro Definition Documentation . . . 273
15.33.2.1 Code . . . 273
15.33.2.2 Code . . . 273
15.33.3 Variable Documentation . . . 273
15.33.3.1 SLASH . . . 273
15.33.3.2 NL . . . 273
15.34 fb-doc_parser.bi . . . 273
15.35 src/Files.md File Reference . . . 277
15.36 src/Files.md . . . 277
15.37 src/Installation.md File Reference . . . 278
15.38 src/Installation.md . . . 278
15.39 src/Introduction.md File Reference . . . 280
15.40 src/Introduction.md . . . 280
15.41 src/Options.md File Reference . . . 282
15.42 src/Options.md . . . 282
15.43 src/Plugin.bas File Reference . . . 288
15.43.1 Detailed Description . . . 289
15.43.2 Function Documentation . . . 289
15.43.2.1 dll_declare . . . 289
15.43.2.2 dll_function . . . 290
15.43.2.3 dll_enum . . . 291
15.43.2.4 dll_union . . . 291
15.43.2.5 dll_class . . . 291
15.43.2.6 dll_define . . . 291
15.43.2.7 dll_include . . . 291
15.43.2.8 dll_init . . . 291
15.43.2.9 dll_error . . . 292

15.43.2.10 dll_empty . . . 292
15.43.2.11 dll_exit . . . 292
15.43.2.12 dll_CTOR . . . 292
15.43.2.13 dll_DTOR . . . 292
15.43.2.14 EmitterInit . . . 292
15.44 Plugin.bas . . . 292
15.45 src/Tables.md File Reference . . . 294
15.46 src/Tables.md . . . 294
15.47 src/Usage.md File Reference . . . 298
15.48 src/Usage.md . . . 298

Index **303**

Chapter 1

fb-doc

fb-doc is a tool for generating documentation from and for FreeBasic source code. Rather than creating the documentation output directly, fb-doc is designed to close the gap between FreeBasic (= FB) and C syntax to allow for using any C / C++ documentation tool-chain. Also it supports the creation of documentational comments in the source and it can get extended via external plugins.

Find more information in the tutorial pages

- Introduction
- Compiling and Installing
- How To Use
- Tables
- Options in Detail
- Emitters in Detail
- Files and Folders
- Examples
- Extending fb-doc
- Further Development

or at the world wide web:

- en: http://www.freebasic.net/forum/viewtopic.php?f=8&t=19810
- de: http://www.freebasic-portal.de/downloads/ressourcencompiler/fb-doc-229.html

1.1 fb-doc licence (GPLv3):

1.2 Acknowledgements

Thanks go to:

- Dimitri van Heesch (author of Doxygen)
- Chris Lyttle, Dan Mueth, Stefan Kost (authors of gtk-doc)
- the FreeBasic developers (a great modern programming language with fun-factor)
- AGS (from http://www.freebasic.net/forum) for testing and bug reporting

Chapter 2

Further Development

In combination with Doxygen fb-doc is a powerful documentation system for FreeBasic source code. The Doxygen support for FreeBasic is much better than for any other BASIC dialect (ie like VB6). But there's still some room for optimization, for Doxygen support as well as for other features, like:

- additional emitters to support further documentation systems or other tools
- a hash table for callees to speed up the execution
- currently fb-doc only works with clean FB code. When one of the used standard keywords get #UNDEFined, grazy things may happen.
- ...

Feel free to post your ideas, bug reports, wishes or patches, either to the project page at

- http://www.freebasic.net/forum/viewtopic.php?f=8&t=19810

or by email to

- see section fb-doc licence (GPLv3):

Chapter 3

Emitters in Detail

3.1 C Source

Emitter	C_Source
`-a` *or* `--asterix`	prepend "* " to each line in multi line comment
`-c` *or* `--cstyle`	emit real C types, #`include "*.c" "*.h"`
`-t` *or* `--tree`	follow source tree #`INCLUDES`

This emitter translates its input in to the intermediate format, usually used by the C back-end. It's not a real compiler, just the documentation relevant constructs get emitted. This is declarations and a few further constructs.

Construct	Keywords
variables	`VAR DIM CONST COMMON EXTERN STATIC`
blocks	`ENUN UNION TYPE CLASS`
forward declarations	`TYPE TYPE alias`
functions	`SUB FUNCTION PORPERTY CONSTRUCTOR DESTRUCTOR`
macros	#`DEFINE` #`MACRO`

It doesn't handle SCOPE nor NAMESPACE blocks yet.

The output contains translated C source code and the special comments for the FB source. Everything is at the same line number and in the original order.

The translation can either contain type declarations in the mangeled (FB-like) style or real types in C syntax. The first is default and the second is set by option `--cstyle`. This option also influences the translation of `TYPE`s (`class{public:` *or* `typedef struct{`), `CONSTRUCTOR`s and `DESTRUCTOR`s (`name::name()` *or* `void()`) and the names in #`include` statements (original *or* suffix `".c" ".h"`).

Option `--asterix` makes fb-doc to start each line in a special multi line comment block by the characters '* '. This is mandatory for the gtk-doc back-end. Adding these characters to the FB source code has two downsides: it blows up the code size and these characters complicate automatic formating of the context in an editor.

The emitter is designed to be used in standard mode (Doxygen filter) and in mode `--file-mode` to generate the intermediate format for the back-end.

Furthermore the output can be helpful when you intend to write a library in FB and later use this library in a C project. This emitter can translate the FB headers to real C headers by using option `--cstyle`. You just have to translate the #`define` lines manually.

3.2 GtkDoc Templates

Emitter	GtkDocTemplates
-a *or* –asterix	

This emitter is designed to generate templates for documenting FB code with gtk-doc tool chain (back-end). It generates output containing the all original code. Documentation relevant parts of the code get prepended by a template for gtk-doc. This template contains the symbol names extracted from the source code and `FIXME` marks for the individual text fragments.

The emitter is designed to be used in mode `--geany-mode`. Installed as a custom command in Geany IDE it receives selected source code and returns it with one or more the templates added.

It doesn't really make sense to use this emitter in modes `--file-mode`, `--list-mode` or `--syntax-mode`, since their output files are used for other context.

Further usage may be generating templates for a complete file

3.3 Doxygen Templages

Emitter	DoxygenTemplates
-a *or* –asterix	no effect
-	

This emitter is designed to

The emitter is designed to be used in mode `--`

It doesn't really make sense to use this emitter in modes `--file-mode`, `--list-mode` or `--syntax-mode`, since their output files are used for other context.

Further usage may be

3.4 List of Function Names

Emitter	FunctionNames
`-a` *or* `--asterix`	no effect
`-c` *or* `--cstyle`	no effect
`-t` *or* `--tree`	follow source tree #`INCLUDES`

This emitter outputs the function names from FB source code. It emitts a list with one name per line for `SUB`s, `FUNCTION`s and `PROPERTY`s. In case of an UDT member the UDT name also gets emitted.

It is designed to generate the list of function names file *fb-doc.lfn* in mode `--list_mode`. This file is used for caller / callee graphs and gets read by the emitter `C_Source`. In this mode the output in the file starts with a header line and the list begins in the second line. The list contains line ends in UNIX style (no caridge return, just a line feed). This must not be changed!

In other run modes no header line gets emitted. The list starts with the first function name found in the input. There is no separator between the output from different input files. But when used in mode `--file-mode` the output is one file per input file.

3.5 Syntax Hightlighting

Emitter	SyntaxHighLighting

-a *or* --asterix	no effect
-c *or* --cstyle	no effect
-t *or* --tree	follow source tree #INCLUDES

This emitter outputs the the original source code, wherein the FB syntax is highlighted by enclosing tags. In mode --syntax-mode three different tag formats can get generated

file type	tags in --syntax-mode
Html (default)	CODE
LaTeX	\textcolor{...}{CODE}
XML	<highlight class="...">CODE</highlight>

In other run modes the emitter generates Html output.

Each line starts with a line number. All input gets separated in to these categories (replacing the '...' in the above tag format examples)

style class	used for
line	normal code no special highlighting
keyword	FB keywords like FOR, NEXT, DECLARE, ...
keywordtype	inbuild types like UBYTE, INTEGER, ... and also BYREF, AS, ...
preprocessor	preprocessor statements like #if, #INCLUDE, ...
stringliteral	string literals like "Hello world"
comment	comments (multi line /' ... '/ or single line ' ...)
lineno	style class for the line number (starting at 1)

Doxygen supports a further style class named keywordflow, meant to be used for keywords like WHILE ... WEND, DO ...LOOP and so on. This doesn't work for FB code since some keywords are used in different meanings (ie FOR ... next and also open(... FOR input)). So fb-doc doesn't use this style class.

This emitter is designed to be used in mode --syntax_mode to repair the syntax hightlighting in Doxygen output. In this mode fb-doc reads the original Doxygen output files and extracts the links to the documentation. The header and footer of the original file get copied to the new file and the listing section get filled with fresh code, including the highlighting tags and also the links from the original Doxygen output.

In other modes like --geany-mode or --list-mode the emitter works in its default mode. Its output contains plain Html code using the above mentioned style tags.

Line numbers start at one, in mode --file-mode for each file. In other modes when the output is generated from several input files, the line numbers increase continuously.

To use the output in any Html page it needs a header (or file) defining the above mentioned style classes. As an example check any Doxygen output (ie the Html tree of this documentation and the file *fb-doc.css*).

Chapter 4

Examples

Here're some examples for using fb-doc and its features.

4.1 Command Line Interface

In a first glance you can execute fb-doc on its own source code in folder *src* to learn about its usage. Extract the archive *fb-doc.zip* to any folder and compile (`fbc -w all fb-doc.bas`). Then you can immediately test the binary in that folder.

Note

The following examples are for UNIX like systems. Omit the leading `./` when you're on other systems.

1. To output the version info and help text execute in a terminal

   ```
   ./fb-doc --version
   ./fb-doc --help
   ```

2. To see the intermediate format for a certain file (fb-doc.bas in this case) execute

   ```
   ./fb-doc fb-doc.bas
   ```

3. Instead of watching the output in the terminal you can write it to a file using

   ```
   ./fb-doc fb-doc.bas > fb-doc.c
   ```

   and compare both files. The file *fb-doc.c* contains just a part of the original code (variable declarations, some special comments, #INCLUDEs and an empty function `main()`). Each context is at the same line number.

4. To test the output of the emitter `SyntaxHighLighting` execute

   ```
   ./fb-doc --emitter "SyntaxHighLighting" fb-doc.bas > fb-doc.html
   ```

   and the new file *fb-doc.html* contains Html code with highlighting tags for that FB source file. (You won't see highlighted code when you load this document in to a browser, because the class declaratinons for the highlighting tags are missing.)

5. Instead of option `--emitter` you can use the short form `-e`. And instead of a single file name you can use a file pattern. Here we test the emitter `FunctionNames` with terminal output

   ```
   ./fb-doc -e "FunctionNames" "*.bi"
   ```

   and get the name `null_emitter` as single output line, which is the only function declared in the code files matching this pattern.

6. When we use `"fb-doc.bas"` as file specification we'll get empty output since this file contains no function declaration. But when we add option `--tree` fb-doc follows each #INCLUDE statement in the source tree and generates a list of all function names.

   ```
   ./fb-doc -e "FunctionNames" --tree "fb-doc.bas"
   ```

The list in the terminal output also contains some messages about the files scanned during the tree run.

7. When you pipe the output to a file, the messages will be shown in the terminal and the file contains the names list only.

   ```
   ./fb-doc -e "FunctionNames" -t "fb-doc.bas" > fb-doc.lfn
   ```

   This list includes the function names of the fb-doc source tree, but it doesn't contain the function names from the file `Plugin.bas`.

8. To generate a list of all function names you can specify several file patterns and collect the function names of all source files in in a single file, like

   ```
   ./fb-doc -e "FunctionNames"  "*.bas" "*.bi" > fb-doc.lfn
   ```

   Option `--recursiv` will make fb-doc searching in subfolders also.

9. Since this list is often needed to generate caller / callee graphs in Doxygen output, fb-doc has the special run mode List Mode (-l) to create it. Use option `--list-mode` (short form `-l`) to activate it and execute fb-doc near the Doxyfile to get the file at the right place

   ```
   cd ../doc
   ./src/fb-doc -l
   ```

   fb-doc scans the *Doxyfile* for the `INPUT` path and the `RECURSIV` setting, uses patterns `"*.bas" "*.bi"` to scan for input files and writes the output to the file *fb-doc.lfn* in the current folder (overriding an existing, if any, without warning)

10. When you installed *Doxygen* and *GraphViz* you're ready to generate your first documentation from FB source. Just execute in the folder *doc*

    ```
    doxygen
    ./src/fb-doc -s
    ```

    The first command generates a lot of Doxygen messages (depending on the settings in *Doxyfile*) and will run some seconds. It should end with message '`*** Doxygen has finished`'. The second command makes fb-doc run in Syntax Mode (-s) and repair the listing sections in the Doxygen output files. After all you should have a new folder *html* (in the folder *doc*) containing your personal version of this documentation. Test it by loading `..doc/html/index.html` in to your browser.

Under the bottom line 1. to 8. are informal examples. You need steps 9. and 10. for your personal project. It should have

- well prepared FB source files (supported by `--geany-mode "DoxygenTemplates"`) and
- a matching Doxyfile (using fb-doc as input filter)

to auto-generate the documentation by executing

```
cd ../doc
fbdoc -l
doxygen
fbdoc -s
```

Note

This conclusion assumes a complete installation of *fb-doc*, *Doxygen* and *GraphViz*.

4.2 Comments (in general)

The C back-ends work on an intermediate format. This format contains comments exported from the FB source. fb-doc doesn't export all comments, since the programmer may want to have private comments as well (ie ToDo marks). On the other hand fb-doc provides a way to export context directly to the intermediate format, to do tricky things with the back-end in use.

To export a comment to the intermediate format it needs to be marked as a special comment by starting its context with a magic character

- `*` for comments (exported as C comment)
- `&` for direct exports (exported as C source)

Note

In these examples the ends of the FreeBasic multi line comments are fakes, because the straight single quote mark cannot be used in the source (Geany lexer gets confused). Instead an acute accent is used. Correct them when you intend to copy / paste the code.

1. A line end comment in the FB source like

```
'* a line end comment
```

gets emitted as

```
/ /! a line end comment
```

2. And a multi line comment in FB source like

```
/'*
Some
lines of
comment '/
```

gets emitted as

```
/**
Some
lines of
comment * /
```

or — when option `--asterix` is set — as

```
/**
* Some
* lines of
* comment * /
```

3. Direct exports are only available for line end comments like

```
'&/ / a line end comment in C syntax

'&/*
'&Some
'&lines of
'&C comment
'&* /
```

get emitted as

```
/ / a line end comment in C syntax

/*
Some
lines of
C comment
* /
```

4. Normal FreeBasic comments (without magic character) wont be emitted.

```
' a line end comment

/'
A block
of
comment
'/
```

They're only visible in the FB source. The intermediate format contains empty lines in that case.

4.3 Variables Declaration

A varible declaration in a FreeBasic source code is in human readable words (making it easy to understand the code) and may look like

```
CONST VarName AS ZSTRING PTR
```

This couldn't get parsed by the C back-end, since this expects the short C syntax like

```
const char* VarName;
```

where

- `const` is a C keyword,
- `char` is the name of an inbuild C type and the
- `*` is similar to the FB keyword `PTR`.

The type declaration `char*` can get replaced by any other type. Rather than checking each type declaration, the parsers in the back-ends just scan the type names and use them to build the documentation.

fb-doc makes use of this flexibility. It creates new C types named similar to the FB syntax (without any declaration). As new type name all FB keywords get mangled in to a single word, separated by underscore characters. The C back-end interpretes this word as a type name and generates documentation output similar to the FB source code. The above example gets in the intermediate (C-like) format

```
CONST_ZSTRING_PTR VarName;
```

and will be used as *CONST_ZSTRING_PTR VarName* in the documentation as well. In this syntax the FB user can identify the type as an FB declaration and easily understand the documentation. See Intermediate Format for more examples.

Note

The type name is allways in front of the symbol name, this order cannot get swapped as in FB syntax.

fb-doc can also generate real C types. In this case the above example gets

```
const char* VarName;
```

and this syntax can be used directly by a C compiler. This can be useful to auto-generate C headers from FB source, ie to bind a library written in FB in to a C project.

4.4 Templates

In an early stage of the project development you'll start to add documentation comments to your source code. When targeting the Doxygen back-end this is an easy task. Just write the comment before or behind the relevant statement in the source code.

It's different for gtk-doc. Each and every documentation comment needs a list of the symbols the comment referes to. It's a boring job to copy the symbol names from the source code to the documentation comment. fb-doc can support this process.

Note

The source code in this section contains verbatim blocks instead of code blocks, because Doxygen miss-interpretes the `\brief` as keyword (allthough it's inside the code block).

4.4.1 Gtk-doc

Note

This example is based on a system wide installation of fb-doc and a setting as Geany custom command with option `--geany-mode`. (See Geany IDE Installation for details.)

This example is about documenting a function and its parameter list. We load our source code in to Geany IDE, select the function declaration (the code in the following box) and send it to the fb-doc custom command:

```
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR
```

fb-doc extracts the names of the function and its parameters, generates a matching template and returns the customized template and the original code. The entries for the individual documentation texts are marked with the text `FIXME`:

```
/'*
goo_axis_new:
@Parent: FIXME
@Back: FIXME
@Text: FIXME
@Modus: FIXME
@...: FIXME

FIXME

Returns: FIXME
'/
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR
```

Now we complete the documentation comment by filling the entries marked by the `FIXME`s, and we end-up with something like:

```
/'*
goo_axis_new:
@Parent: the parent item, or %NULL. If a parent is specified, it
         will assume ownership of the item, and the item will
         automatically be freed when it is removed from the
         parent. Otherwise call g_object_unref() to free it.
  @Back: the background box to connect the axis to (a
         #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
         Note: to set the axis position and size, the properties
         #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
         #GooCanvasItemSimple:width and
         #GooCanvasItemSimple:height will be red (and therefore
         must be set in the background box item).
  @Text: the label text for the axis
 @Modus: the position and type (like %GOO_AXIS_SOUTH or
         %GOO_GRIDAXIS_SOUTH, ...)
   @...: optional pairs of property names and values, and a
         terminating %NULL.

Creates a new axis item.

Returns: (transfer full): a new axis item.
'/
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
```

```
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR
```

Finally we translate all FB sources to C source using (defaults: output path = `../doc/c_src`, file specification = `"*.bas" "*.bi"`)

```
fbdoc --file-mode --asterix
```

The corresponding *.c* file looks like

```
/**
* goo_axis_new:
* @Parent: the parent item, or %NULL. If a parent is specified, it
*          will assume ownership of the item, and the item will
*          automatically be freed when it is removed from the
*          parent. Otherwise call g_object_unref() to free it.
*   @Back: the background box to connect the axis to (a
*          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
*          Note: to set the axis position and size, the properties
*          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
*          #GooCanvasItemSimple:width and
*          #GooCanvasItemSimple:height will be red (and therefore
*          must be set in the background box item).
*   @Text: the label text for the axis
*  @Modus: the position and type (like %GOO_AXIS_SOUTH or
*          %GOO_GRIDAXIS_SOUTH, ...)
*    @...: optional pairs of property names and values, and a
*          terminating %NULL.
*
* Creates a new axis item.
*
* Returns: (transfer full): a new axis item.
* */
FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (
BYVAL_AS_GooCanvasItem_PTR Parent,
BYVAL_AS_GooCanvasItem_PTR Back,
BYVAL_AS_gchar_PTR Text,
BYVAL_AS_GooAxisType Modus,
...) {

};
```

This C-like output can be used as input for the gtk-doc tool-chain to generate the desired documentation output. Execute the tools on the files in the *../doc/c_src* folder.

Or you use option `--cstyle` to generate output with types in real C syntax

```
fbdoc --file-mode --asterix --cstyle
```

and the corresponding *.c* file looks like

```
/**
* goo_axis_new:
* @Parent: the parent item, or %NULL. If a parent is specified, it
*          will assume ownership of the item, and the item will
*          automatically be freed when it is removed from the
*          parent. Otherwise call g_object_unref() to free it.
*   @Back: the background box to connect the axis to (a
*          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
*          Note: to set the axis position and size, the properties
*          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
*          #GooCanvasItemSimple:width and
*          #GooCanvasItemSimple:height will be red (and therefore
*          must be set in the background box item).
*   @Text: the label text for the axis
*  @Modus: the position and type (like %GOO_AXIS_SOUTH or
```

```
*          %GOO_GRIDAXIS_SOUTH, ...)
*    @...: optional pairs of property names and values, and a
*          terminating %NULL.
*
* Creates a new axis item.
*
* Returns: (transfer full): a new axis item.
* */
GooAxis* goo_axis_new (
GooCanvasItem* Parent,
GooCanvasItem* Back,
gchar* Text,
GooAxisType Modus,
...) {

};
```

4.4.2 Doxygen

Note

This example is based on a system wide installation of fb-doc and a setting as Geany custom command with option `--geany-mode "DoxygenTemplates"`. (See Compiling and Installing for details.)

This example is about documenting a function and its parameter list, similar to the previous one. We load our code in to Geany IDE, select some source like the following code and send it to the fb-doc custom command:

```
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR
```

fb-doc extracts the names and returns the customized template, followed by the original code:

```
/'* \fn FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (BYVAL_AS_GooCanvasItem_PTR Parent, BYVAL_AS_GooCanvasItem_
\param Parent FIXME
\param Back FIXME
\param Text FIXME
\param Modus FIXME
\param ... FIXME

\brief FIXME

\returns FIXME

FIXME

'/
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR
```

The first line is the declaration of our function in C syntax — without the character ; at the end and prepended by the keyword \'fn'. This line enables Doxygen to read the documentation comment anywhere in its input, so the documentations comment needs not to be in front of the matching function. Since we don't wonna move this block (it's advantageous to place the documentation text near to the source code) we can remove this line and then complete the documentation comment by filling the `FIXME` entries. We end-up with something like:

```
/'* \brief Creates a new axis item.
\param Parent the parent item, or %NULL. If a parent is specified, it
         will assume ownership of the item, and the item will
```

```
         automatically be freed when it is removed from the
         parent. Otherwise call g_object_unref() to free it.
\param Back the background box to connect the axis to (a
         #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
         Note: to set the axis position and size, the properties
         #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
         #GooCanvasItemSimple:width and
         #GooCanvasItemSimple:height will be red (and therefore
         must be set in the background box item).
\param Text the label text for the axis
\param Modus the position and type (like %GOO_AXIS_SOUTH or
         %GOO_GRIDAXIS_SOUTH, ...)
\param ... optional pairs of property names and values, and a
         terminating %NULL.

\returns a new axis item.

A detailed desription of the function may follow here ...

'/
FUNCTION goo_axis_new CDECL( _
  BYVAL Parent AS GooCanvasItem PTR, _
  BYVAL Back AS GooCanvasItem PTR, _
  BYVAL Text AS gchar PTR, _
  BYVAL Modus AS GooAxisType, _
  ...) AS GooAxis PTR

' ...
END FUNCTION
```

Using Doxygen we don't need to generate intermediate C source code in files. Instead this code gets directly piped by fb-doc acting as a filter, so we usually don't see it. Being curious we may execute in a terminal `fbdoc XYZ.bas` (replace XYZ by your file name) and we'll see output (which is the Doxygen input) like

```
/**
\param Parent the parent item, or %NULL. If a parent is specified, it
         will assume ownership of the item, and the item will
         automatically be freed when it is removed from the
         parent. Otherwise call g_object_unref() to free it.
\param Back the background box to connect the axis to (a
         #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
         Note: to set the axis position and size, the properties
         #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
         #GooCanvasItemSimple:width and
         #GooCanvasItemSimple:height will be red (and therefore
         must be set in the background box item).
\param Text the label text for the axis
\param Modus the position and type (like %GOO_AXIS_SOUTH or
         %GOO_GRIDAXIS_SOUTH, ...)
\param ... optional pairs of property names and values, and a
         terminating %NULL.

\brief Creates a new axis item.

\returns: a new axis item.

The detailed desription of the function follows here ...

*/
FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (
BYVAL_AS_GooCanvasItem_PTR Parent,
BYVAL_AS_GooCanvasItem_PTR Back,
BYVAL_AS_gchar_PTR Text,
BYVAL_AS_GooAxisType Modus,
...) {

};
```

Note

This output is also visible in the source code browser before the listings get repaired by executing `fbdoc -s` in the folder *doc*.

Chapter 5

Extending fb-doc

fb-doc is an open source project and everybody is able to customize the source code to his personal needs. Forking the project has the downside that changes have to be redone in case of an update of the original source.

fb-doc offers an alternative by its included interface for external emitters (= plugins). An external emitter is a shared library that gets loaded at runtime. The fb-doc project and the plugin are independend projects. In worst case the external modul has to get recompiled when the used headers were changed in fb-doc.

5.1 Internals

Before we can extend fb-doc we should first understand some internals on how it works and how it serves the handlers in the EmitterIF.

fb-doc loads the input in to a parsers buffer Parser::Buf. Then the parser scans this context in two phases:

1. searching end of statements (new line or colon) and checking the start of the next statement
2. if this is a relevant construct it gets tokenized by Parser::tokenize()

So only relevant constructs get checked in detail. The tokenizer generates a list of tokens containing three INTEGER values for each entry:

1. the token type Parser::ParserTokens
2. the start of the token in the buffer (zero based) and
3. its length in bytes

The complete construct gets tokenized. This is a single line in case of an `#INCLUDE`, but also may be a bunch of lines in case of a block (`#MACRO ENUM TYPE SUB ...`). The tokenizer uses just a subset of the FB keywords, check function Parser::getToken() for details. All other words from the source code get `TOK_WORD` in the list. White spaces (like space, tab, caridge return, vertical tab, ...) and also other context without a specifier in Parser::ParserTokens like numerical expressions get no entry in the token list. If this stuff is needed for an emitter, the emitter has to do the parsing itself.

Then the parser checks the token list at the beginning of the construct.

In case of syntax errors the handler EmitterIF::Error_() gets called and the parser drops the construct. It's up to the emitter how and where to output the error message or to parse the buggy construct anyway.

In case of correct syntax the parser calls the matching emitter handler in the EmitterIF. Priviously, during the syntax check, some pointers are set to the tokens in the list. The handler can use this pointers and the tokenlist to generate its output.

These most important pointers are set always

- Parser::StaTok the first (start) token of the construct (*ie* `TOK_SUB` *in* `SUB name ...` *and* `TOK_DECL` *in* `DECLARE SUB name ...`)
- Parser::EndTok the last token in a construct (`TOK_EOS`, *check* Parser::Buf[TokEnd[1]] *if it's a colon or a new line*)
- Parser::Tk The current position of the parser.

More than 15 further pointers are used in the parsers check, see Parser for details. But not all pointers are set or reset for each construct. Find further information the source code of these parser functions:

Function	Constructs
Parser::FUNCTION_()	`SUB FUNCTION PROPERTY OPERERATOR CONSTRUTOR DESTRUCTOR`
Parser::VAR_()	`DIM REDIM VAR CONST COMMON EXTERN EXPORT STATIC`
Parser::TYPE_()	`TYPE CLASS`
Parser::UNION_()	`UNION`
Parser::ENUM_()	`ENUM`
Parser::DECLARE_()	`DECLARE`
Parser::DEFINE_()	`#DEFINE`
Parser::MACRO_()	`#MACRO`
Parser::INCLUDE_()	`#INCLUDE`

Some functions are available to parse advanced constructs like lists of several variable declarations:

- Parser::parseListNam evaluate a list of names
- Parser::parseListNamTyp evaluate a list of declarations (name and type)
- Parser::parseListPara evaluate a parameter list
- Parser::parseBlockEnum

As paratmeter these helpers get a function pointer to an EmitFunc handler. This handler gets called on each member of the list. Each member is of the same type here (parameter, enumerator, variable, ...).

It's a little more complex when you have to handle a `TYPE` or `UNION` block. The helper function for this to parse is:

- Parser::parseBlockTyUn

and the passed handler function has to deal with several kinds of context (like variable declarations or 'function declarations). Also these blocks may contain further nested blocks that may or may not get parsed recursivly (depending on the purpose of the emitter). The handler has to deal with all this kind of stuff. Find examples in any EmitterIF::Clas_ handler, ie in the handler c_Block().

5.2 Control Flow

When loading an external emitter plugin fb-doc first calls the function EmitterInit to receive the pointer to the external EmitterIF. This gets done right after parsing the command line and before a Parser UDT is created.

???

5.3 Example Plugin.bas

Enough of theory, let's switch to practise. The fb-doc parser calls emitter functions via the emitter interface EmitterIF, so it can easy get extended by new emitters providing additional features (ie support further C tools like *source navigator*).

The *src* folder of archive *fb-doc.zip* contains a file *Plugin.bas*. This file is an example source code for an external emitter modul and contains this further information:

```
This emitter generates a list of the function names called via the
emitter interface. So when you input some source to fb-doc and use
this emitter, the output is a list of the functions called by the
parser for this input.

Before you can use this emitter, you have to compile it first, using
the command

\code fbc -dylib Plugin.bas \endcode

The result is a binary called

  - libdll_emitter.so  (LINUX)
  - libdll_emitter.dll (windows)

There's no way to compile or use an external emitter on DOS since
DOS doesn't support dynamic linked libraries.

To use this emitter in fb-doc set its name (without the suffix .bas)
just as an internal name (,so don't name your customized emitter
similar to an internal emitter name).

Ie. the emitter output for the context of this file can be viewed in
the terminal by
~~~{.sh}
./fb-doc --emitter "Plugin" Plugin.bas
~~~
(LINUX example) and will generate the following output:

\verbatim
DLL_INIT
DLL_INCLUDE
DLL_INCLUDE
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_EXIT
\endverbatim
```

Note

The entry in EmitterIF::Nam of an external emitter plugin is not used in fb-doc, its free for any usage.

Chapter 6

Files and Folders

fb-doc currently uses two external files. It tries to load the context when needed and continues without the context when the files aren't present. fb-doc searches for the files in the EXEPATH (the folder of the fb-doc binary).

6.1 ../doc/c_src

This folder gets created by fb-doc to write the file output. It's used to collect the translations from the *C_Source* emitter. The back-end read its input from here.

6.2 fb-doc.lfn

The suffix lfn means list of function names. This file is used in the C_Source emitter to create pseudo function calls in a function body. It gets created by the ListOfFunction emitter when using (option ´–list-mode) and it isn't used in Geany mode (option–geany-mode‘, emitters GtkDocTemplates and DoxygenTemplates).

Allthough it's usually created by fb-doc, it also can get manipulated by any text editor.

The file contains the names of all functions in the source code. Member functions of a UDT are listed as they are named in the function body (ie TypeName.FunctionName). The first line is empty. Further lines contain the names, one in each line, including the dot in case of a member function. The lines **must** be separated by a single line feed character (LINUX line end = 'CHR(10)'). So be careful when we edit on DOS / windows (ie use Geany and switch to LF line ends).

This file is needed when running as a filter for Doxygen to create the caller and callees graphs. Therefor the bodys of the function have to include the function calls. fb-doc checks all words in a function body against the known function names. Any match gets emitted as a pseudo function call (with empty parameter list).

Since in this mode Doxygen serves the file names in any order and fb-doc gets started new for each file, it doesn't know about a function declaration from file A when working on file B. Therefor the function names have to be served from an external source. They get red from this file at each program start.

The file offers an additional way to control the context of the caller and callees graphs. Only functions listed in this file gets included in the C source (and in the graphs).

When this file isn't present fb-doc continues execution without any message and the function bodys of the pseudo C code are empty. This means no caller or callees graphs will be created.

6.3 External Moduls (∗.so / ∗.dll)

fb-doc can use external emitter modules. These are executable binaries, compiled and linked as a library for dynamic linkage, either by using the FreeBasic compiler or any other compiler / linker combination. These files may

have any name and get the system specific file name extension (= suffix) .so on LINUX or .dll on windows.

Such files aren't necessary for basic usage of fb-doc. They are used for external emitters to extend the fb-doc features and get loaded on demand (option `--emitter`) at run-time, see Extending fb-doc for details. So you may have one or more files with one of these suffixes in your working directory.

Chapter 7

Compiling and Installing

Before you can start, you've to build your personal binary of fb-doc. It gets shipped as a FreeBasic source tree. Find the code in the folder *src* in your *fb-doc.zip* archive. To create your binary you need to install the FreeBasic compiler fbc for your operating system. Then just extract the *fb-doc* folder to any place on your hard disk, change to the folder *src* and compile the main file *fb-doc.bas*. Ie load the file in to Geany IDE and choose menu *Build -> Compile*. Or call the Freebasic compiler at the command line (in the extracted folder *src*)

```
cd .../fb-doc/src
fbc -w all fb-doc.bas
```

This creates an executable binary named

- *fb-doc* (on UNIX-like systems) or
- *fb-doc.exe* (on other systems).

That's all you need to get started. Now you can use fb-doc and check its features, ie translate FB code to intermediate C format or generate templates.

Therefor you don't need a complex installation. Just place the compiled binary in any of your source folders and execute it there (see How To Use for details). Or you test your fb-doc executable on its own source, find some examples in Command Line Interface.

To generate a real documentation — as mentioned in the Introduction, fb-doc is not a complete documentation generator — you have to install an additional back-end for C source, like

- gtk-doc
- Doxygen

Download one of these packages for your operating system and follow the installation instruction. Both can get extended by using further tools. Consider to install them too. (Ie the package GraphViz is used to generate the caller / callee graphs in this documentation.)

The first back-end is a good choise if you like to document a library API in the GNOME documentation style. In all other cases — especially when you have to document a program or application — the later is the better choise. Doxygen is a more modern tool, it's easier to handle, comes with a GUI front-end and has more features.

When you intend to use fb-doc in a regular basis and call it from different folders for several projects, it's best to install fb-doc on your system, as specified in one of the two following sections.

7.1 UNIX Installation

It's recommended to create a link to the executable and move this link to a folder in the system PATH. Using a link allows

- recompiling of the source (ie for a new release) and also
- using the fresh executable from any directory.

This can be done by executing in a terminal

```
cd .../fb-doc/src
cp -l fb-doc ~/bin/fbdoc
```

Assuming that the folder ~/bin is in your PATH you can now execute fb-doc in each folder by calling the link. Try

```
cd ~
fbdoc --version
```

and you should see the version information text in the terminal. This solution works on your personal user account. When you need a system wide installation that allows all users to access fb-doc, then you need admin privileges to place the link in a similar system folder, ie in like

```
cd .../fb-doc/src
sudo cp -l fb-doc /usr/bin/fbdoc
```

Note

Replace ... / by a proper path on your system.

7.2 DOS / windows Installation

DOS doesn't support links and on the other system links don't work at the terminal (AFAIK). So it's best to use a batch file to call the fb-doc executable.

Here's the recommended way

1. create a folder C:\bin (or use an existing folder for batch files in you PATH)
2. ensure that this folder is in the system PATH
3. create a file named fbdoc.bat in this path with the following context

```
C:\YOUR\PATH\TO\fb-doc\src\fb-doc.exe %*
```

Then test the installation by executing

```
cd \
fbdoc --version
```

and you should see the version information text in the terminal.

Note

Replace C:\YOUR\PATH\TO by a proper path on your system.

7.3 Geany IDE Installation

fb-doc can be used as a filter (= custom command) for Geany IDE. Using this feature, we can send the current selection (a text fragment) to the filter and receive the filtered output as replacement for the selected text. From a user point of view it looks like paste a new text block in to a previously selected section.

To get this running choose menu item *Edit->Format->Send Selection to->Set Custom Commands* and click on *Add* to get a new item. Then type

```
fbdoc --geany-mode
```

to use the default emitter for gtk-doc templates or select the emitter for Doxygen templates by

```
fbdoc --geany-mode "DoxygenTemplates"
```

To test the installation open a new editor window. Now right-click on an empty line and choose menu item *Format->Send Selection to->YOUR_NEW_COMMAND_ITEM*. The following text should appear (assuming you're using the DoxygenTemplates setting)

```
/'* \file FIXME
\brief FIXME

FIXME

'/
```

This is the output from the function EmitterIF::Empty_() of the `DoxygenTemplates` emitter. Such a comment block is used to start any Doxygen input file.

If you don't get this text nor any other message in the editor then check the *status* message tab (in the notebook window at the bottom) to get a hint for solving the problem.

Note

Instead of the right-click menu try also keybindings — `<Crtl>-[1 | 2 | 3]` is default. Find further details in `Geany documentation`.
This setting only works when fb-doc is installed in your system `PATH` (see previous sections for details). If you don't want to install fb-doc, you have to use the full path and the original name of the executable (ie like `/home/me/proj/fb-doc/src/fb-doc -g Doxy`).

Chapter 8

Introduction

It's state-of-the-art in software development to write and edit the documentation context inside the source code. The programmer(s) can adapt the documentation on each improvement or bug fix at one place. All work is done in the source file(s), the documentation is placed in special comments in the source code so that the compiler doesn't see them. Beside the compiler, which extracts information for the CPU, an additional tool-chain is used to parse the special comments and the related source code to build the documentation, avoiding redundand data in separate files and keeping comments to a minimal size.

Powerful tool-chains (back-ends) exist for several programming languages (like C) to generate output in different formats (ie like *html*, *pdf*, *man-pages* and others). But previous to fb-doc there is nothing for FreeBasic source code yet.

Rather than being a complete tool-chain fb-doc is designed as a bridge to existing C back-ends, since it's a lot of work to build and test such a complete tool-chain for several output formats and keep it up to date. Instead fb-doc creates an intermediate format (similar to the FB source code, but with C-like syntax) that can be used with existing, well developed and tested C documenting back-ends. fb-doc has been tested with

- `gtk-doc`
- `Doxygen` The later is used for this documentation.

The steps to generate a well documented project are

1. generate source code, compile and test it
2. add documentation comments inside the source code and keep them up to date
3. run fb-doc on the FreeBasic source code to build a C-like intermediate format
4. run the C-back-end on the intemediate format to build the documentation output in one or more output formats (ie *html*, *pdf*, *manpage*, ...)

This intermediate format doesn't contain complete C source and cannot get compiled by a C compiler. Instead it just contains all information the lexical scanner of the back-end needs to build the documentation output. Therefor fb-doc itself doesn't parse the complete FreeBasic source code. Instead it also acts as a lexical scanner to extract just the necessary information.

Note

That's why fb-doc shouldn't be executed on buggy source (be prepared to face fancy output in that case).

8.1 fb-doc

fb-doc is a multi functional tool, supporting the complete process of documenting FB source code. You can use it at the command line and manually control its operations by individual command line options. It's also designed to operate as an automated tool to act

- in a *Makefile*,
- as a filter for *Doxygen* or
- as a custom command for *Geany*.

Several run modes control where to get input from and where to write output at. Several emitters are available do generate different kinds of output. Each run mode has its default emitter, but an individual emitter can be selected instead. A plugin system makes it possible to extend fb-doc by external emitters.

All this gets controlled by the command line interface. It may be a bit overhelming at the beginning. It's recommended to concentrate on the examples to get startet.

8.2 Documentation Context

The documentation context gets build from the FB source code and its surrounding comments. fb-doc exports both in a C-like intermediate format as input for the used back-end.

Not all comments get exported. Only comments starting with a magic character are used to build the documentation, see Comments (in general) for details.

Also, not all FB source code gets exported. The output is reduced to the minimal set of constructs to build the documentation, see C Source for details. This makes the execution of the tool-chain faster and avoids confusion due to the foreign language for the lexical scanner.

This constructs and their comments get emitted at the same line number in the intermediate format to enable the tool-chain referencing to a certain line.

8.3 About this Text

And finally some words about this documentation (the text you're currently reading). It is generated by Doxygen back-end with fb-doc filter. Find examples for a lot of topics by studying the files in the folder *src* of the fb-doc package. Beside this you may want to check the files in the folder *doc*. configurations for Doxygen to extract this pages. A good start may be to use these files and create your customized fb-doc documentation and play around with the parameters, see Command Line Interface.

To be honest: in some cases this documentation may be a bit overloaded and serve more information than necessary. But one of the reasons for creating it is to demonstrate the features of fb-doc and Doxygen. Therefor not all possibilities are used to reduce the output to the essentials (see the Doxygen manual for details).

Note

It's not under the scope of this documentation to describe the usage of any (or all possible) tool-chain(s). Please refer to the respective manual(s) for further information.

Chapter 9

Options in Detail

This page contains more detailed informations about the options.

fb-doc gets controlled by the command line, containing elements of this types (may be none, one or more than one)

- options (starting with '–' or '––')
    1. mode options
    2. operational options
    3. option parameters (separated by white space)
- file specifications (in any combination of)
    1. a single file name
    2. a list of single file names (separated by white spaces)
    3. a file pattern
    4. a list of file patterns (separated by white spaces)

Note

An empty command line defaults to option `--help`.

The options get evaluated in the given order. When two options are used with similar effects, fb-doc stops with and error message. Example (both options are contrary run mode options).

```
prompt$ fb-doc --list-mode --geany-mode
fb-doc: Invalid command line (multiple run modes)
```

The options `--help` and `--version` are dominat. They stop further command line parsing and get executed immediately.

Since a mode option sets its standard emitter (and overrides any previous emitter settings), it's best practise to set options in the following order

```
fb-doc [mode options] [operational options] <file specifications>
```

Note

The options information in the following summary tables belong to the standard emitters. ???

9.1 Mode Options

The standard mode is the Doxygen-Filter mode. fb-doc operates in this mode when no other mode option is set. The only exeption is an empty command line (no file specification). In this case fb-doc switches to Help (-h).

9.1.1 Default Mode

Summary	Standard Mode (Doxygen Filter)
Input	file (path and name)
Output	STDOUT (context depends on emitter)
Default Emitter	C_Source (in FB style)
File Spec	FreeBasic source code (like ∗.bas;∗.bi – no default)
Further Options	-a -c -e -r -t (depends on emitter)
Ignored Options	-o

In standard mode fb-doc reads input from files and sends output to STDOUT. The default emitter is `C_Source` in default mode. The file specification may contain a single file name, a path and afile name, a file pattern or a path and a file pattern. Several file specifications can be used in any combination, separated by a white space character. The emitter output for each file gets collected and send in a single stream to STDOUT.

The standard mode is designed to be used as a filter for Doxygen. In that case a single file name (including a path, if any) is passed at the command line. fb-doc operates on this file and emits its output to STDOUT pipe.

Furthermore this mode is helpful to test the fb-doc output (ie when developing a new emitter). Just specify your emitter and an input file like

```
fb-doc --emitter "Plugin" fb-doc.bas
```

and fb-doc operates on the file fb-doc.bas and pipes the output of the *Plugin* emitter to the terminal.

Or you can use this mode to generate a C-header for a library, collecting the context of several FB headers in one C header. While it has some advantages to use several header files during the development, further users of the library may prefer to have only one file. This can be done by scanning the source files and piping STDOUT to a file

```
fb-doc --emitter C_Source --c_style *.bi > MyLib_SingleHeader.h
```

When your headers need a certain order you may use the option `--tree` and specify the start files instead of the pattern.

Or you can generate a file for external documentation (outside the source code). Let fb-doc collect all relevant symbols by using the `--tree` option as in the example above. Or extract symbols in a per file basis, like

```
fb-doc --emitter "DoxygenTemplates" MyFile_abc.bas > MyFile_abc.txt
```

Then edit all `FIXME` entries in MyFile_abc.txt and include the file in to your Doxygen file tree.

9.1.2 File Mode (-f)

Summary	File Mode
Input	file (path and name)
Output	file (.bi -> .h, .bas -> .c)
Default Emitter	C_Source (in FB style)
File Spec	FreeBasic source code (default: ∗.bas ∗.bi)
Further Options	-a -c -e -o (defaults to ../doc/c_src) -r -t
Ignored Options	(depends on emitter)

The file mode is designed to read input from certain files and to write output to certain files. For each input file an output file gets created in the output folder specified by option `--outpath`. The type of the output file depends on the emitter in use:

Emitter	∗.bas Type	∗.bi Type	default path
C_Source	–> ∗.c	–> ∗.h	../doc/c_src
SyntaxHighLighting	–> ∗.bas.html	–> ∗.bi.html	../doc/fb_html
GtkDocTemplates	–> ∗.bas.gtmp	–> ∗.bi.gtmp	../doc/src

DoxygenTemplates	–> *.bas.dtmp	–> *.bi.dtmp	../doc/src
FunctionNames	–> *.bas.lfn	–> *.bi.lfn	../doc/src
external plugin	–> *.bas.ext	–> *.bi.ext	../doc/src

fb-doc creates a new folder if the output folder doesn't exist. Also higher level directories get created if not existend. When an input file comes from a subfolder, a similar subfolder gets created in the output folder. When a subfolder is beyond the current path (ie like `../../../src` form an #INCLUDE statement in a source file) the .. part(s) of the path get skipped and fb-doc creates subfolders for the rest of the path.

Note

fb-doc never writes above the output folder.
Existing files in the output folder get overriden without warning.

fb-doc uses the above mentioned default paths if option `--outpath` is not used. To write in to the current directory use option `--outpath .` (path name is a dot).

The most common use is to bridge to the gtk-doc back-end. Usually a set of FB source files get translated in to similar C source files using the C_Source emitter. gtk-doc then operates on that C files to generate the desired output. So usually fb-doc gets executed in the source folder like

```
fb-doc --file-mode --asterix --cstyle --recursiv
```

to operate on all FB files (using default `*.bas *.bi`) in the current folder and its subfolders to generate a similar source tree in the default output folder *../doc/c_src*. Then switch to this folder and start the gtk-doc tool chain there.

Furthermore you can add a file pattern (or multiple) to operate on selected files (ie "??_view*.b*"). Or we use this run mode in a Makefile to update a set of C sources like

```
fb-doc --file-mode --asterix --cstyle $@
```

Note

The renaming of the output files as in the above table is inbuild in fb-doc source code and cannot get adapted by command line settings.

9.1.3 Geany Mode (-g)

Summary	**Mode**
Input	STDIN
Output	STDOUT
Default Emitter	GtkDocTemplates
File Spec	none
Further Options	-e
Ignored Options	-a -c -o -r -t (depends on emitter)

The geany mode is designed to generate templates for the documentational comments in the source code, see section Geany IDE Installation for details. Usually a code section gets selected in the editor and then sent to fb-doc via STDIN pipe. fb-doc extracts the relevant symbols and generates a matching comment block for this piece of code. Both, the comment block and the original code block, get returned to geany and replaces the previously selected block.

From a geany user point of view it looks like adding a block in to the source, but this block contains individual informations related to the previously selected block.

Geany mode is also usefull for testing purposes because the output to STDOUT doesn't change any files in your source tree. Just edit the custom command in Geany settings to change to the desired emitter. This option allows to specify an emitter name directly after the option (since geany mode doesn't need any file specifications) but you can also use option `--emitter` in this mode. The output can either be tested in the geany editor, where you can easy select the stuff for fb-doc. Or you can pipe an input stream to fb-doc at the command line. The following example lists all function names from file test.bas in the terminal

```
fb-doc --geany "FunctionNames" < test.bas
```

Note

The output may contain error messages from fb-doc. Those are send to STDERR and mixed in to the output by the terminal. That's different when directing the STDOUT stream to a file. In that case the error messages are shown in the terminal while the file contains pure fb-doc output.

Furthermore you can use this mode to collect output from several files in to a single one. The following example collects the C-translations of the declarations from two FB headers in the file C_Header.h

```
fb-doc --geany "C_Source" --c-style < file1.bi > C_Header.h
fb-doc --geany "C_Source" --c-style < file2.bi >> C_Header.h
```

9.1.4 List Mode (-l)

Summary	Mode
Input	file (path from Doxyfile, name = ∗.bas;∗.bi)
Output	fb-doc.lfn
Default Emitter	ListOfFunction
File Spec	Doxyfile (but also ∗.bas;∗.bi)
Further Options	-a -c -e -r -t
Ignored Options	-o (depends on emitter)

The list mode is designed to generate a list of function names and write this list to the file fb-doc.lfn, one function name per line. Later, this file is used to generate pseudo function calls in the function bodies by the C_Source emitter, making Doxygen being able to generate caller / callee graphs.

Usually the mode is used in the `../doc` folder near the Doxyfile. fb-doc reads the `INPUT` path from the Doxyfile and scans all FB source files, to extract the function names in a single file `fb-doc.lfn`, which gets written near the Doxyfile (where fb-doc can find it later on when called by Doxygen as input filter). Example

```
cd ../doc
fb-doc --list-mode
```

In this example no input file name is specified and fb-doc uses the default name `Doxyfile`. Also it's possible to specify any outher file name and a prepending path (ie like `../doc/fb-doc.Doxyfile`). But if the file specification contains a pattern (characters ∗ or ?) or the extension is one of `.bas` or `.bi`, then fb-doc skips the reading of the Doxyfile and operates on the FB source file(s) directly.

9.1.5 Syntax Mode (-s)

Summary	Syntax Mode
Input	files (pathes and names from Doxyfile)
Output	files
Default Emitter	SyntaxHighLighting
File Spec	Doxyfile
Further Options	-e -o
Ignored Options	-a -c -r -t (depends on emitter)

In syntax mode fb-doc opens the specified Doxyfile and evaluates the path to the FB source and the output types and their folders. Then it scans the output folders for files containing listings, operating on Html, LaTeX and XML output.

These listing files get replaced by a version with the original header and footer and inbetween a newly created listing section with appropriate FB syntax highlighting. All links found in the original files get transfered to the new file.

The syntax mode is designed to correct syntax highlighting in the Doxygen output files. Since the paths get evaluated from the Doxyfile, no options (additional to `--syntax-mode`) are neccessary. By default fb-doc read the file named "Doxyfile", but its possible to specify (one or more) individual file name(s).

Since this mode operates on specific Doxygen output files, it makes no sense to use it with an alternative emitter (but fb-doc doesn't block this scenario).

9.1.6 Help (-h)

Summary	Help Mode
Input	none
Output	STDOUT (version text)
Default Emitter	none
File Spec	none
Further Options	none
Ignored Options	none

This option makes fb-doc to output the help text and stop. The help text contains a brief summary of all available option and some examples for usage. It should help experienced users to remember some information. (It's not mentioned to be a complete documentation.)

Note

In help mode all other options have no effect.

9.1.7 Version (-v)

Summary	Version Mode
Input	none
Output	STDOUT (help text)
Default Emitter	none
File Spec	none
Further Options	none
Ignored Options	none

This option makes fb-doc to output the version information and stop. The version information contains the source code version number, the date and time of compilation and the used operating system for the fbc compiler.

Note

In version mode all other options have no effect.

9.2 Operational Options

Operational options control the fb-doc operations or the output of an emitter. In special cases they may have no effect, depending on the used combination of run mode and emitter.

9.2.1 Asterix (-a)

-a	**--asterix**
Parameter	none
Run Modi	all
Emitters	`C_Source`

This options makes the emitter `C_Source` to output an asterix character and a white space (like `"* "`) at the start of each line in a special multi line comment block, see Gtk-doc for further examples.

This special format is used in gtk-doc (must have) and can also be used in Doxygen (no advantage, slows down execution).

Editing and formating a special comment block with these line starts is complicated and slow. The asterix character max get mixed up with the text when using line wrapping functions. Therefor fb-doc offers this feature to edit the documentation context in plain text and add the special format only in the output for the back-end.

9.2.2 CStyle (-c)

-c	--cstyle
Parameter	none
Run Modi	all
Emitters	C_Source, DoxygenTemplates

This option makes fb-doc to emit real C types instead of the FB-like mangled type names. It also infuences the translation of TYPE blocks and #INCLUDE lines, see Intermediate Format for examples.

The standard output of the C_Source emitter is optimized for best matching documentation. Therefor the types in the source code get fantasy names, similar to the FB keywords declaring them. Use this option to switch form FB sytle to real C type names.

9.2.3 Emitter (-e)

-e	--emitter
Parameter	Emitter name
Run Modi	all

This option makes fb-doc to use an alternative emitter and overrides the run mode default emitter setting. A parameter must follow this option (separated by a white space), specifying the emitter name. The parameter may be surrounded by quotes (single or double), they get removed befor further operation.

First, fb-doc searches in the list of internal emitter names (see Inbuild Emitters). This search gets done non-case-sensitive and partial-matching. Meaning you need not type the complete emitter name nor use the right letter cases. Ie *d*, *Dox* or *"DOXY"* all match the full emitter name *DoxygenTemplates*.

In case of no match in the internal emitter names fb-doc tries to load an external emitter with the specified name. In this case the emitter name must exactly match the base file name of the FB source code used to build the plugin module. Ie when the plugin was compiled by

```
fbc -dylib Plugin.bas
```

the parameter *emitter name* must be Plugin as in

```
fbdoc -e "Plugin"
```

Note

On UNIX-like systems file names are case-sensitive.
For external emitter plugins specify the base name of the source file (not the context of the string EmitterIF::-Nam).

9.2.4 Outpath (-o)

-o	--outpath
Parameter	path to folder
Run Modi	--file-mode, --list-mode
Emitters	all

This option is used to specify the path for the fb-doc file output. It works for the above mentioned run modi, wherein fb-doc generates new file output (but not for --syntax-mode where fb-doc replaces files generated by Doxygen).

The parameter may be either a relative path starting at the current directory (where fb-doc is executed) or an absolute path (starting with "/" on UNIX-like systems or with a drive letter and a colon an other systems).

fb-doc writes the file output in the specified directory. The directory gets created first, if it doesn't exist. Also all higher level directories get created if not existing yet.

In `--list-mode` the file *fb-doc.lfn* gets created in that directory, overriding an existed without warning (if any).

In `--file-mode` more than one file may get created, depending on the specified input file(s) or pattern(s). In case of option `--recursiv` or `--tree` also subdirectories may get created in the putpath directory and its subfolder(s).

Note

fb-doc writes files in to the outpath folder and may create subfolders in it. But fb-doc never performs any changes in directories above the outpath.

9.2.5 Recursiv (-r)

-r	--recursiv
Parameter	none
Run Modi	all
Emitters	all

This options makes fb-doc scanning for input files in the working folder and in its subfolders. It takes only into effect when the input file specification is a file pattern (or a list of patterns).

When the file pattern has no path, the current folder is the working folder. Otherwise fb-doc scans the path specified before the pattern and its subfolders.

In the following example the current folder is *doc* and fb-doc scans the working folder *src* and its subfolders

```
cd myProj/doc
fbdoc -r "../src/*.bas" "../src/*.bi"
```

Note

The option has no effect when a single file name (or a list of names) is specified (it's only used for patterns).
When fb-doc operates on a Doxyfiles (*in* `--list-mode` *or* '–syntax-mode) the setting of its *RECURSIV* parameter overrides this option.
It has no effect when running in `--geany-mode`.
On LINUX systems usually the shell (bash) expands the file patterns and sends a list of single names to fb-doc, so this option has no effect until you enclose the file pattern by quotation marks (like in the example above).

9.2.6 Tree (-t)

-t	--tree
Parameter	none
Run Modi	all
Emitters	all

This option makes fb-doc to follow the source code tree. That is, all #`INCLUDE` statements will be evaluated and fb-doc operates on this files as if they were specified as input files on the command line.

Note

This only works with files in the source code tree. Standard header files (ie like `"crt/string.bi"`) wont be found since fb-doc doesn't know the standard FreeBasic include path.
For this option to work the emitter must provide a handler for EmitterIF::Incl_() in which the parsing of the new files get started. Not all emitters do support this.

9.3 File Specifications

A file specification is used to determine one or more file(s) for fb-doc file input (so not for `--geany-mode` where fb-doc gets input from STDIN). Each entry at the command line that is neither an option nor an option parameter gets recognized as file specification and added to the list Options::InFiles for further operation.

A file specification contains an (optional) path to the working directory and either a concrete file name or a file pattern. Examples

File Specification	Description
`abc.bas`	the file `abc.bas` in the current directory
`../doc/Doxyfile`	the file named `Doxyfile` in the working folder `../doc`
`*.bas`	all files matching the pattern `*.bas` in the current folder
`../src/*.bi`	all files matching the pattern `*.bi` in the working folder `../src`

Whereat the current directory is the folder fb-doc was executed in. And the working folder is the directory specified by the path part of the file specification. The later can either be a relative path (as in the examples above) or an absolute path (starting with "/" on UNIX-like systems or with a drive letter and a colon on other systems).

File specifications can be used in any combinations. Use white spaces to separate them. When a path or a file name contains a white space it must be enclosed by (single or double) quotes. Generally it's benefiting to enclose any file specification in qoutes, especially on UNIX-like systems whereat unquoted pattern get expanded in the shell and fb-doc receives a list of names instead of the pattern (option `--recursiv` doesn't work in that case).

fb-doc uses default file specifications if none is set in the command line. The default depends on the specified run mode:

Run Mode	Default File Specification
default mode	`*.bas *.bi`
`--file-mode`	`*.bas *.bi`
`--list-mode`	`Doxyfile`
`--syntax-mode`	`Doxyfile`

Chapter 10

Tables

This page contains some information on fb-doc in table format.

10.1 Overview

fb-doc is a multi functional tool. While several Input / Output specify where to get input from and where to write output to, several Inbuild Emitters generate different kind of output. Here's a table of all run modes against the inbuild emitters ("DEF" = default configuration, "+" = useful combination, "-" = combination not useful).

Emitter \ Run Mode	default	`--file-mode`	`--list-mode`	`--syntax-mod`	`--geany-mode`
C_Source	DEF	DEF	-	-	+
GtkDoc-Templates	+	+	-	-	DEF
Doxygen-Templates	+	+	-	-	+
FunctionNames	+	+	DEF	-	+
SyntaxHigh-Lighting	+	+	-	DEF	+

10.2 Options

The following table contains an overview of all fb-doc options. An option either starts by a minus character followed by a single character (short form) or by two minus characters followed by a word or pair of words (long form). Both forms have the same meaning.

Some options expect an additional parameter. Each further word in the command line (without the leading '-' character) gets interpreted as a file name or pattern, see Options in Detail for details.

Option	Parameter	Description
Asterix (-a)	none	The C_Source emitter generates lines in a multi line comment block with leading '∗ ' characters (gtk-doc style).
CStyle (-c)	none	The C_Source emitter generates types in real C syntax (instead of FB styled pseudo C syntax). Also used in emitter `DoxygenTemplates`.

Emitter (-e)	Emitter name	Customized emitter selection. fb-doc compares the parameter with the names of the internal emitters. In case of no match it tries to find and load an external with this name.
File Mode (-f)	none	Read FB source files and write output to similar named files (overriding existend, if any).
Geany Mode (-g)	Emitter name	Read input from STDIN and write to STDOUT. The parameter is optional.
Help (-h)	none	Print the help text and stop.
List Mode (-l)	none	Read FB source files and write output to file *fb-doc.lfn*.
Outpath (-o)	path	Set folder for file output.
Recursiv (-r)	none	Scan for input files in the working folder and all subfolders. Works only in combination with file patterns.
Syntax Mode (-s)	none	Scan a Doxyfile (or multiple) for its output and fix syntax highlighting.
Tree (-t)	none	Follow the #INCLUDE statements in the source tree (if possible – not available in Geany Mode (-g)).
Version (-v)	none	Print the version information and stop.

Note

An empty command line makes fb-doc to output the help text (as if option `--help` was given).

10.3 Inbuild Emitters

fb-doc has five inbuild emitters to generate different kinds of output. All Mode Options set their default emitter, but you can customize selection by option `--emitter` (`-e`) afterwards. Also this option is used to choose an external emitter (plugin), see Extending fb-doc for details.

Note

Only one emitter can be in use at a time.

Name	Default in Mode	Function
C Source	`--file-mode` and default (Doxy-Filter)	Translated FB source in intermediate format (C-like syntax). Use FB-like typenames (default) or real C syntax (option `--cstyle`). Prepend asterix in comment blocks (option `--asterix`).

GtkDoc Templates	`--geany-mode`	Emit source code and prepend documentation relevant constructs by templates for gtk-doc. Prefered usage in `--geany-mode`.
Doxygen Templages	none	Emit source code and prepend documentation relevant constructs by templates for Doxygen. Prefered usage in `--geany-mode`.
List of Function Names	`--list-mode`	Emit a list of all function names. Prefered usage in `--list-mode` to generate the file *fb-doc.lfn*.
Syntax Hightlighting	`--syntax-mode`	Emit all source code surrounded by syntax highlighting tags. In `--syntax-mode` for Doxygen output in *∗.html, ∗.tex* and *∗.xml* files. In other modes *html* tags only.

10.4 Input / Output

Some options control the execution of fb-doc and the streams of input and output data. The following table shows the relations between run mode and input / output targets, See Mode Options for details.

Option	Input	Output	Default Emitter
none (Doxy-Filter-Mode)	one file	standard out (STDOUT)	C_Source
`-f` *or* `--file-mode`	one file, a list of file names, a file pattern, a list of file patterns or all source files	*∗.c* and *∗.h* files	C_Source
`-g` *or* `--geany-mode`	standard in (STDIN)	standard out (STDOUT)	GtkDocTemplates
`-l` *or* `--list-mode`	files matching a pattern or all source files	file *fb-doc.lfn*	FunctionNames
`-h` *or* `--help`	none	standard out (STDOUT)	none
`-v` *or* `--version`	none	standard out (STDOUT)	none

Note

the default mode is the Doxygen filter mode, since Doxygen doesn't allow to send further options (directly). Input and output channels are fixed to the listed options. But you can apply changes, ie by specifying an other emitter with option `--emitter` or input more than one file by walking through the source tree with option `--tree`, ...
The modi `--help` (`-h`) and `--version` (`-v`) are not related to any FB source. Instead of using an emitter they create fixed output on standard output channel (STDOUT).
When more then one run mode is specified in the command line, the latest is the dominant.

10.5 File Specifications

Any text in the command line not matching an option or its parameter gets interpreted as a file name or file pattern. They get added to a queue and fb-doc executes this queue in the given order. Depending on the file name and the run mode fb-doc operates in different ways:

Option (run-mode)	no name	exact name	pattern
default	print help text	load file, emit output to STDOUT	load all files matching the pattern, emit output to STDOUT
-f *or* –file-mode	load all ∗.*bas* and ∗.*bi* files, emit output for each file to a ∗.*c* or ∗.*h* file	load this one file, scan the source tree and load each #INCLUDE file, emit output for each file to a ∗.*c* or ∗.*h* file	load all files matching the pattern, emit output for each file to a ∗.*c* or ∗.*h* file
-g *or* --geany-mode	ignored	ignored	ignored
-l *or* --list-mode	load all ∗.*bas* and ∗. *bi* files, emit output to one file named *fb-doc.lfn*	load this one file, scan the source tree and load each #INCLUDE file, emit output to one file named *fb-doc.lfn*	load all files matching the pattern (or multiple patterns), emit output to one file named *fb-doc.lfn*

Note

A file name may be prepended by a relative or an absolute path.
File names or paths including a space character have to be enclosed by single or double quotes.
Multiple file names get separated by a space character or by a ; character.
You can list several file names and patterns in the command line.
A file name cannot start with a minus character and cannot include single or double quote characters.
On UNIX like systems usually the shell (bash) expands the file patterns (fb-doc doesn't see the pattern, but gets the files list instead, so option --recursive doesn't work – enclose the pattern by single or double quotes to hinder that).

10.6 Intermediate Format

The main trick to make a C parsers work on FB source is to transform the declarations for variables and functions. FB syntax is full of keywords for better human readability. Most of them are unknown for the C parser (fbc-0.90 has more than 400 keyword, in C it's less than 50).

And the C parsers expect just a single word for a type declaration. But when we pack all FB information in a single word type name, the C tool can handle this syntax and build the documentation output.

Therefor the FB keywords get mangled in to one single word, separated by an underscore character. The following table contains some examples. This mangling can get suppressed by option --cstyle, to get real C types emitted. There's also some influences on the emitted TYPEs and file names of include files.

FB source	default	--cstyle
DIM AS INTEGER Nam	INTEGER Nam	int Nam
DIM Nam AS INTEGER	INTEGER Nam	int Nam
CONST Nam AS INTEGER	CONST_AS_INTEGER Nam	const int Nam
Extern Nam AS INTEGER	Extern_AS_INTEGER Nam	extern int Nam
DECLARE PROPERTY Cu.Tok() AS INTEGER PTR	PROPERTY_AS_INTEGER_PTR Cu.Tok()	int *Cu.Tok(void)
BYVAL Export_ AS EmitFunc	BYVAL_AS_EmitFunc Export_	EmitFunc *Export_
byref Z as byte	byref_as_byte Z	char *Z
byval Z as byte	byval_as_byte Z	char Z
TYPE Udt ... END TYPE	class Udt{ ... };	typedef Udt{ ... };
#INCLUDE ONCE "abc.bi"	#include "abc.bi"	#include "abc.h"
#INCLUDE ONCE "xyz.bas"	#include "xyz.bas"	#include "xyz.c"

10.7 Files

Here's an overview on the important files in the archive *fb-doc.zip*.

The folder *src* contains the source code files of fb-doc and some additional text for this documentation in the file *Tutorial.bi*.

fb-doc/src	Function
fb-doc.bas	The main source code to compile (#INCLUDEs other files).
Tutorial.bi	The text of this Tutorial (no FB source in there).
Plugin.bas	The source code for an external emitter (example).
other ∗.bas ∗.bi	The source code.

The folder *doc* is to build this documentation by executing Doxygen in this folder, check its manual for details. If you want caller / callee graphs in the documentation you have to install the *dot* tool from the *GraphViz* package and you have to execute `fb-doc -l` before you start doxygen. For repairing the syntax highlighting in the Doxygen output run `fb-doc -s` afterwards in this folder.

fb-doc/doc	Function
Doxyfile	A configuration file, controlling the Doxygen and fb-doc operations.
DoxygenLayout.xml	A configuration file, controlling the index of these *html* pages.
fb-doc.lfn	A list of function names generated by executing `fb-doc -l`.

Chapter 11

How To Use

This is a brief introduction in auto-generating documentation extracted from the source code. Therefor you'll need well written source code in correct syntax for the compiler (FB syntax), but also comments matching the syntax of the back-end in use.

So to test fb-doc and to learn about its usage and features you'll need

1. source code to work on (ie the fb-doc code in the `src` folder)
2. an executable of fb-doc (ie compiled by `fbc fb-doc.bas`)
3. a documentation back-end (fb-doc comments are formated for Doxygen)
4. optional a GUI frontend (ie Doxywizard)
5. optional further tools (ie *GraphViz* (graphs), *LaTeX* (pdf), ...)

both, an installation of a C tool-chain and some source code with documentation comments in the tool-chain's syntax.

The easiest way to get started is to use Doxygen and the fb-doc sourc code. In that case you can skip the next section.

11.1 Generating Templates

For Doxygen back-end there's no need for generating templates. It's best practise to write the documenting comment right in front of or just behind the relevant construct. The only exeption is a documenting comment for a function. It contains the names of the variables in the parameter list.

Its different when using gtk-doc tool-chain. Here the documentation context is collected in blocks. Such a block contains the name of the relevant construct (ie a ENUM block) and the names of all its members, followed by its decription. (Doxygen can handle such blocks as well, but it isn't well supported.)

Generating such a comment block is a reasonable amount of word. Each name has to get copied from the source in to the block. fb-doc can do this for us.

When we use Geany IDE, we can use a convenient method by installing fb-doc as Geany custom command (see Geany IDE Installation) and choose the emitter for the tool-chain in use (*GtkDocTemplates* or *DoxygenTemplates*). After loading the source, we select a block of code, send it to fb-doc and we receive the original block, prepended by a customized documentation block. We just need to edit the entries (marked by the text `FIXME`). See Gtk-doc and Doxygen for examples.

We can select a single construct and generate the comment blocks one-by-one. Or we select a bunch of statements and fb-doc inserts the templates inbetween the constructs. In case of a block construct (ENUM / UNION / TYPE) it's advantageous to select the complete block up to the END ... statement, to get all members listed in the

documentation block. In case of a function decalaration the selection need not include the function body, but should contain the complete decalaration.

It's a bit less convenient to auto-generate templates when an other IDE is in use. In that case the documentation blocks can get generated for a complete file, by piping its context to fb-doc. See Piping for details.

11.2 Gtk-doc

When using gtk-doc as back-end the documentation comment blocks are placed before a structure, union, function ... and the names of the members or parameters are listed in the comment block, prepended by a @ character and appended by a colon. See the manual http://developer.gnome.org/gtk-doc-manual/stable/index.html for details. Find an example for a gtk-doc documentation comment in Gtk-doc.

fb-doc helps us by creating such documentation comment blocks (= template). We select a piece of code in Geany and send it to the fb-doc filter. fb-doc creates a template, evaluates the names from our source code and includes them in the template, followed by the text `FIXME`. We just replace this text by the proper description. To use this feature we have to install fb-doc as Geany filter, see Geany IDE Installation for details.

When our documentation texts are done, we create a pseudo C source tree in a separate folder and use the gtk-doc tool chain on this C files. Therefor we start a command line interpreter (shell) and switch to the folder where our FB source is placed. Then, we execute

```
fb-doc --asterix --file-mode
```

This makes fb-doc creating a new folder *../doc/c_src* (if not present) and write pseudo C files similar to our FB source (overriding existing files). ∗.*bas* files get translated to ∗.*c* and ∗.*bi* files get translated to ∗.*h*. Our documentation comments get transfered to this files and some of our FB code gets translated to C syntax. Then we start the gtk-doc tool-chain in the *../doc/c_src* folder as descibed in the above mentioned manual (*gtkdoc-scan*, ...).

gtk-doc works well to document a library API. Especially when it's related to GLib, GTK+ or gnome software. One of the unique features is auto-extracting the properties from the source code (I'll publish an FB tool soon). But it's not the best choise when we want to document a program like fb-doc. In that case we better use Doxygen.

11.3 Doxygen Back-End

For Doxygen we can either use separate comment blocks, similar to the gtk-doc blocks. Doxygen blocks use a different syntax and they don't need to be placed in front of the related construct. fb-doc also supports Doxygen templates, but this way of documenting isn't very common and not well supported yet.

The other case is — a more commen way — to place the documentation directly in front of (or behind the) related construct. Rather than documenting a complete block at once, just the class name or just one member gets documented in one comment. This way is more convenient, we need not extract the names from our source code. We just place the documentation beside the construct and Doxygen picks the names from the source. For that way of documenting there's no need to generate templates. The only execption is a parameter list, where the parameter names have to be listed in the documentation block. See

- the manual http://www.stack.nl/~dimitri/doxygen/manual.html for details.
- Doxygen for an example for a function comment block
- all fb-doc source code as further examples.

For Doxygen we can also create the pseudo C files in the *../doc/c_src* folder, as described in the previous section. But it's more convenient to send the pseudo C code directly to Doxygen without creating intermediate files. This can be done by using fb-doc as a Doxygen filter. (We use Doxygen as if we were working on C source code, fb-doc translates on-the-fly when Doxygen reads the input.)

Doxygen gets controlled by a configuration file. This file contains all options. Here we can specify the fb-doc filter by adapting the following:

First we have to tell Doxygen that it should load FB source files:

```
FILE_PATTERNS          = *.bi \
                         *.bas
```

and second we need to set fb-doc as the filter for these files

```
FILTER_PATTERNS        = *.bas=fbdoc \
                         *.bi=fbdoc
```

Note

Doxygen doesn't allow to send additional options directly to a filter. So if we need further options we can set the name of a batch file here and call fb-doc from the batch file passing further options.

That's it. When we now start doxygen with proper settings for the paths and output format, we get our first auto-created documentation. (Try it with the fb-doc source code.)

To get caller and callees graphs (as in this document) it gets a little tricky. Doxygen needs the function calls inside the function bodies (in the C source). But when fb-doc acts as a Doxygen filter it gets restarted on each file. So it doesn't know a function declaration in file B when working on file A. The names of all the functions must be known to evaluate their calls in a function body and to generate proper pseudo C source. That's why fb-doc reads the names from a file called *fb-doc.lfn* in the current folder.

We needn't generate this file manually, fb-doc can do this for us by executing the command

```
fb-doc --list-mode
```

in our source folder. This writes a new (or overrides an existing) file *fb-doc.lfn* in the current folder. When we don't start Doxygen in our source folder, we have to move the file to the folder where we execute the *doxygen* command. When we use Doxywizzard (the GUI frontend for Doxygen) this is the folder specified under

```
Step1: Specify the working directory from which doxygen will run
```

And we have to specify fb-doc as filter for source files and enable source file filtering

```
FILTER_SOURCE_FILES    = YES

FILTER_SOURCE_PATTERNS = *.bas=fbdoc \
                         *.bi=fbdoc
```

In the next run Doxygen should be able to generate caller and callee graphs in the documentation (if the settings are correct and all tools are available).

But this has a downside: when we want the source files in the documentation (as in this document) the C-like code is used. To get the FB code we have to run Doxygen twice:

1. without source filtering to get the *html* files with FB source
2. with source filtering to get the graphs

Before the second run we save the *html* source files in a separate folder by executing in the Doxygen html output folder something like (example for LINUX OS)

```
mkdir bas
mv *bas_source.html bas
mv *bi_source.html bas
```

and after the second run we restore the FB source files

```
rm *bas_source.html
rm *bi_source.html
cd bas
mv * ..
cd ..
rmdir bas
```

11.4 C Headers

Since fb-doc needs to generate C-like code for the back-end parsers, it can also be useful to create a set of C headers for a library written in FreeBasic. By default the FB types get mangled in to one word (for documentation purposes). Ie a FB parameter *byref Nam as const short* gets the pseudo C code *byref_as_const_short Nam*. This mangling can get suppressed by option `--cstyle` and fb-doc emitts real C types. In that case the example gets *const short∗ Nam* and this code can be used by a C compiler.

So when we need C headers for our FB library we just execute

```
fb-doc --file-mode --cstyle "*.bi"
```

in our source folder and get a set of C translations in ∗.*h* files in the target folder (*../doc/c_src* by default).

Note

fb-doc doesn't handle initializers. You have to check them manually.

11.5 Piping

By default fb-doc reads its input for (one or more) files and sends its output to STDOUT. We can switch to STDIN input by option *–geany-mode* and then use the operating system commands to control the data flow in and from fb-doc.

Ie this is useful when we don't use Geany IDE or when we want to get templates for a complete file or a bunch of files at once. We can pipe the files context to the fb-doc STDIN channel and the fb-doc output to a file.

To get templates for all constructs in an existing file, first we create a copy the file (example for LINUX OS)

```
cp filename.bas filename.txt
```

Then we pipe the copy to fb-doc in geany mode (using emitter *DoxygenTemplates* here). The output gets piped to the original file (omit >*filename.bas* to see the output in the console)

```
fbdoc --geany-mode doxy < filename.txt > filename.bas
```

Finally we may want to delete the intermediate file

```
rm filename.txt
```

To insert templates in all files of an existing project we write a batch file (or FB program) that does these three steps for all our source files.

Chapter 12

Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Doxyfile
: Handle a Doxyfile 55

EmitterIF
: The emitter interface 57

Highlighter
: Class to process the syntax highlighting 65

Options
: Parameters red from the command line 77

Parser
: The parser 92

RepData
: A container for string replacements 131

Chapter 13

File Index

13.1 File List

Here is a list of all files with brief descriptions:

src/fb-doc.bas
: The main source file to compile . . . 148

src/fb-doc.bi
: Header for the main source file . . . 152

src/fb-doc_doxyfile.bas
: The source code for the Doxyfile class . . . 153

src/fb-doc_emit_callees.bas
: Emitter for the file *fb-doc.lfn* . . . 159

src/fb-doc_emit_doxy.bas
: Emitter for Doxygen templates . . . 168

src/fb-doc_emit_gtk.bas
: Emitter for gtk-doc templates . . . 178

src/fb-doc_emit_syntax.bas
: Emitter for repairing the Doxygen syntax highlighting . . . 189

src/fb-doc_emitters.bas
: Default emitter to create pseudo C source, #INCLUDEs the other emitters . . . 228

src/fb-doc_emitters.bi
: Header file for EmitterIF . . . 241

src/fb-doc_options.bas
: The source code for the Options class . . . 243

src/fb-doc_options.bi
: Header file for options . . . 251

src/fb-doc_parser.bas
: Source code for the Parser class . . . 256

src/fb-doc_parser.bi
: Header file for the Parser class . . . 273

src/Plugin.bas
: An example 'Code for an external emitter . . . 292

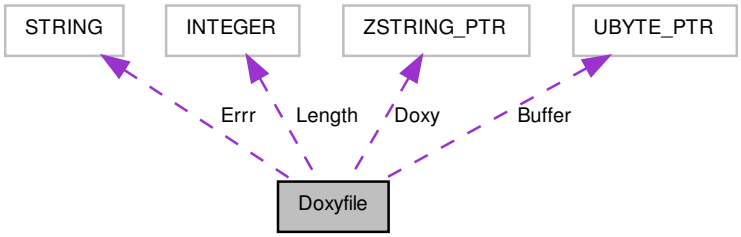
Chapter 14

Class Documentation

14.1 Doxyfile Class Reference

handle a Doxyfile

Collaboration diagram for Doxyfile:



Public Member Functions

- Doxyfile (BYREF_AS_STRING)

  construct reading a Doxyfile
- ~Doxyfile ()

  destructor freeing the memory used
- FUNCTION_AS_STRING Tag (BYREF_AS_STRING)

  find a single line tag in the Doxyfile and return its value

Public Attributes

- INTEGER Length
- UBYTE_PTR Buffer
- ZSTRING_PTR Doxy = @"@"empty"
- STRING Errr

14.1.1 Detailed Description

handle a Doxyfile

Read a Doxyfile and parse its coontext. In case of errors finding or loading the file the Errr variable gets set in the constructor. Its empty on success. Currently only single line tags are supported (no lists).

Definition at line 21 of file fb-doc_doxyfile.bas.

14.1.2 Constructor & Destructor Documentation

14.1.2.1 Doxyfile::Doxyfile (BYREF_AS_STRING *Fnam*)

construct reading a Doxyfile

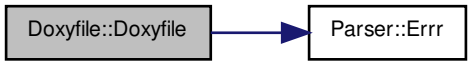
Parameters

Fnam	The (path and) file name

The constructor tries to find and read the Doxyfile. In case of errors the variable Errr contains an error message. Its empty otherwise and the context can get parsed by the function Tag().

Definition at line 41 of file fb-doc_doxyfile.bas.

Here is the call graph for this function:



14.1.2.2 Doxyfile::~Doxyfile ( )

destructor freeing the memory used

The destructor frees the memory allocated in the constructor, if any.

Definition at line 63 of file fb-doc_doxyfile.bas.

14.1.3 Member Function Documentation

14.1.3.1 FUNCTION_AS_STRING Doxyfile::Tag (BYREF_AS_STRING *Su*)

find a single line tag in the Doxyfile and return its value

Parameters

Su	the tag to search for

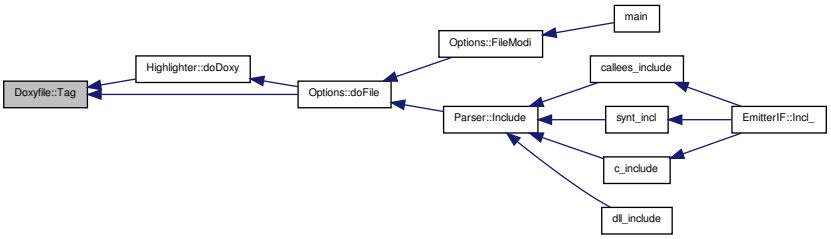
Returns

the tags value (if any))

This function is designed to find a value of an Doxyfile tag. It return the tags context or an empty string if the tag has no value or cannot be found.

Definition at line 76 of file fb-doc_doxyfile.bas.

Here is the caller graph for this function:



14.1.4 Member Data Documentation

14.1.4.1 INTEGER Doxyfile::Length

Definition at line 22 of file fb-doc_doxyfile.bas.

14.1.4.2 UBYTE_PTR Doxyfile::Buffer

Definition at line 23 of file fb-doc_doxyfile.bas.

14.1.4.3 ZSTRING_PTR Doxyfile::Doxy = @"@"empty"

Definition at line 24 of file fb-doc_doxyfile.bas.

14.1.4.4 STRING Doxyfile::Errr

Definition at line 25 of file fb-doc_doxyfile.bas.

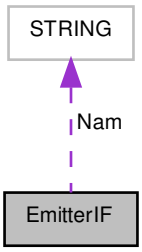
The documentation for this class was generated from the following file:

- src/fb-doc_doxyfile.bas

14.2 EmitterIF Class Reference

The emitter interface.

Collaboration diagram for EmitterIF:



Public Member Functions

- void Init_ (void)

  function called before parsing a source code
- void Decl_ (void)

  emitter for a declaration (VAR / DIM / CONST / COMMON / EXTERN / STATIC)
- void Func_ (void)

  emitter for a function (SUB / FUNCTION / PROPERTY / CONSTRUCTOR / DESTRUCTOR)
- void Enum_ (void)

  emitter for a ENUM block
- void Unio_ (void)

  emitter for a UNION block
- void Clas_ (void)

  emitter for a struct (TYPE / CLASS block)
- void Defi_ (void)

  emitter for a macro (#DEFINE / #MACRO)
- void Incl_ (void)

  emitter for includes (#INCLUDE)
- void Error_ (void)

  emitter for an error message
- void Empty_ (void)

  emitter for an empty Geany block
- void Exit_ (void)

  function called after parsing the source code
- void CTOR_ (void)

  function called at program start-up (once)
- void DTOR_ (void)

  function called at program end (once)

Public Attributes

- STRING Nam = ""

  the emitters name

14.2.1 Detailed Description

The emitter interface.

The emitters interface is a UDT containing function pointers. The Parser calls the matching function in the active emitter after scanning a relevant construct. The emitter function extracts the necessary information from the parser data, formats it as desired and sends it to the output stream.

Only one emitter can be active at a time. Either one of the four inbuild fb-doc emitters or an external emitter plugin can be choosen by option `--emitter`.

The function pointers get initialized with the null_emitter() function (SUB), that creates no output at all. Each emitter modul can replace one or more of the pointers by a customized function to create a specific output.

Since Doxygen doesn't support to generate documentation for such an interface, it cannot create caller or callee graphs for the emitter functions. But we use fb-doc and can work-around this by creating additional C output in form of member function. These functions are unvisible for the FreeBasic compiler, but get emitted to the pseudo C source for the Doxygen back-end and produce the desired output for the documentation.

Definition at line 51 of file fb-doc_emitters.bi.

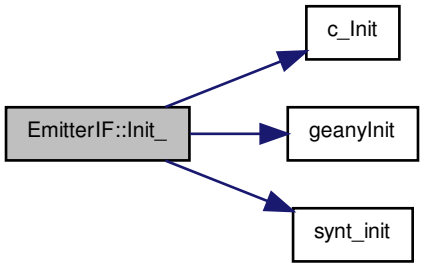
14.2.2 Member Function Documentation

14.2.2.1 void EmitterIF::Init_ (void) [inline]

function called before parsing a source code

Definition at line 72 of file fb-doc_emitters.bi.

Here is the call graph for this function:

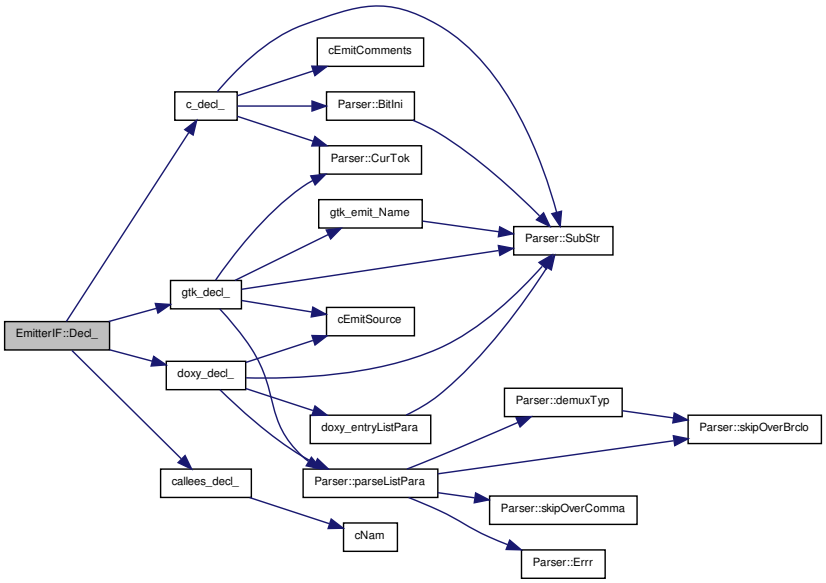


14.2.2.2 void EmitterIF::Decl_ (void) [inline]

emitter for a declaration (VAR / DIM / CONST / COMMON / EXTERN / STATIC)

Definition at line 75 of file fb-doc_emitters.bi.

Here is the call graph for this function:

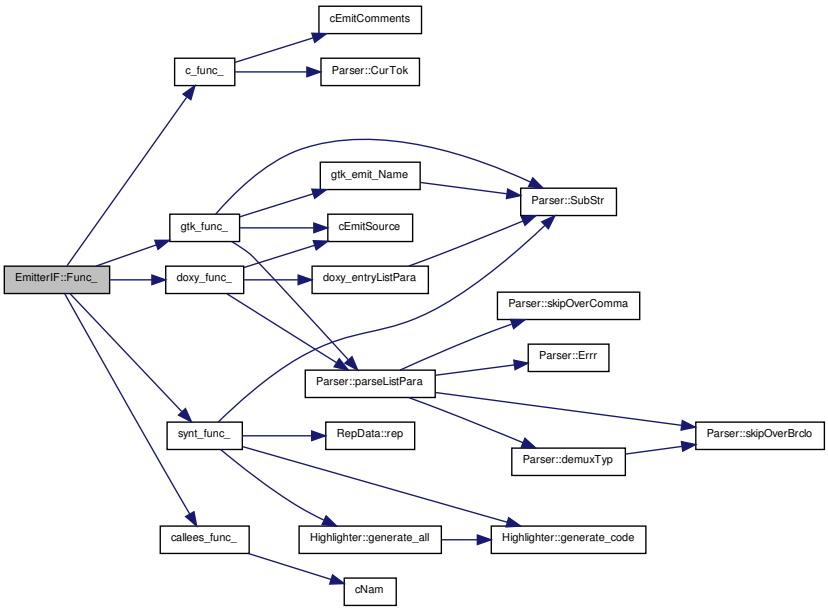


14.2.2.3 void EmitterIF::Func_ (void) [inline]

emitter for a function (SUB / FUNCTION / PROPERTY / CONSTRUCTOR / DESTRUCTOR)

Definition at line 78 of file fb-doc_emitters.bi.

Here is the call graph for this function:

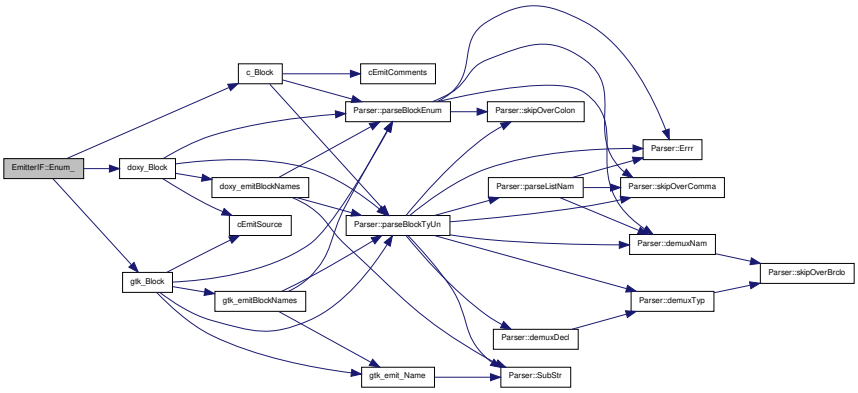


14.2.2.4 void EmitterIF::Enum_ (void) [inline]

emitter for a ENUM block

Definition at line 81 of file fb-doc_emitters.bi.

Here is the call graph for this function:

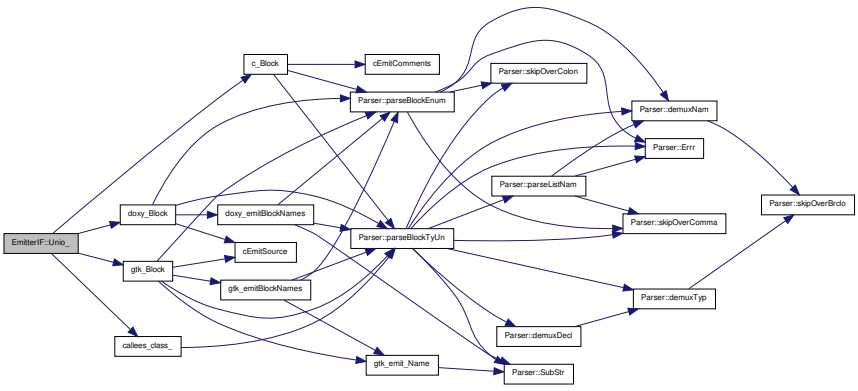


14.2.2.5 void EmitterIF::Unio_ (void) [inline]

emitter for a UNION block

Definition at line 84 of file fb-doc_emitters.bi.

Here is the call graph for this function:

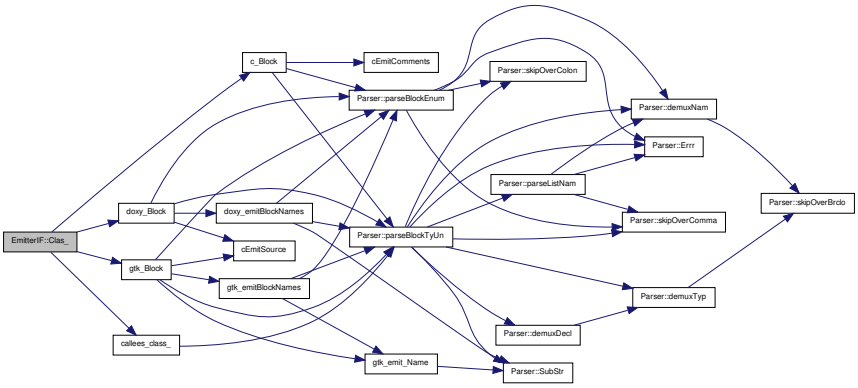


14.2.2.6 void EmitterIF::Clas_ (void) [inline]

emitter for a struct (TYPE / CLASS block)

Definition at line 87 of file fb-doc_emitters.bi.

Here is the call graph for this function:

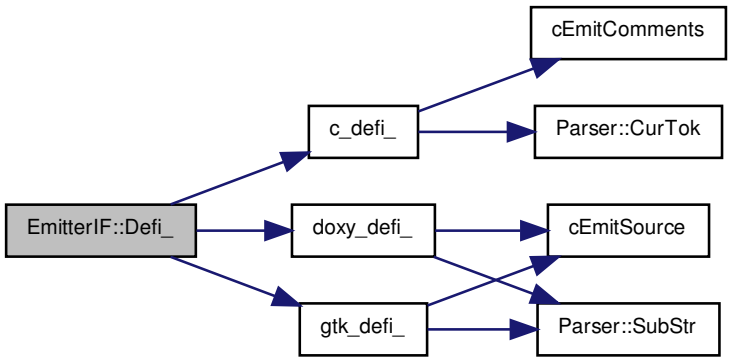


14.2.2.7 void EmitterIF::Defi_ (void) [inline]

emitter for a macro (#DEFINE / #MACRO)

Definition at line 90 of file fb-doc_emitters.bi.

Here is the call graph for this function:

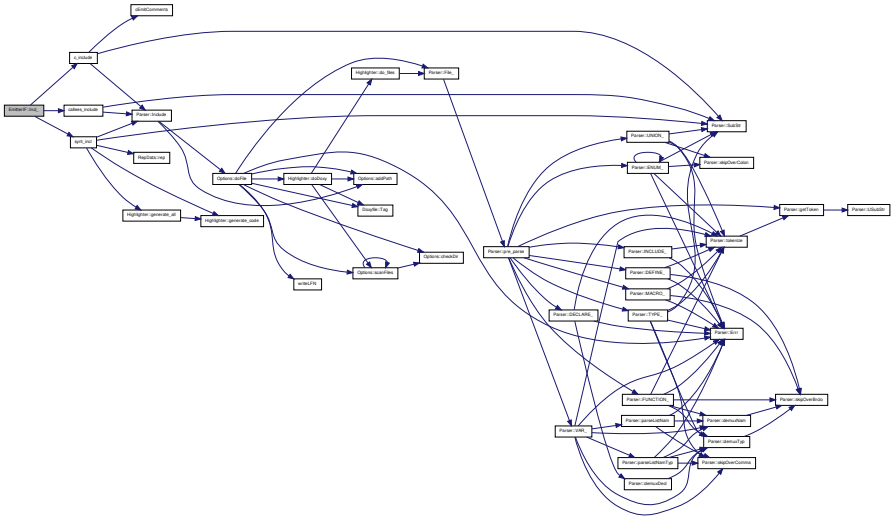


14.2.2.8 void EmitterIF::Incl_ (void) [inline]

emitter for includes (#INCLUDE)

Definition at line 93 of file fb-doc_emitters.bi.

Here is the call graph for this function:

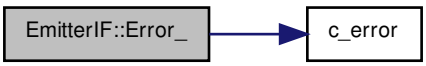


14.2.2.9 void EmitterIF::Error_ (void) [inline]

emitter for an error message

Definition at line 96 of file fb-doc_emitters.bi.

Here is the call graph for this function:

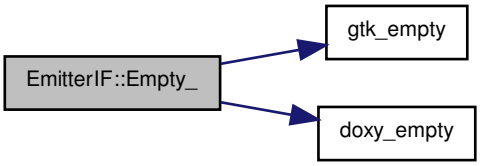


14.2.2.10 void EmitterIF::Empty_ (void) [inline]

emitter for an empty Geany block

Definition at line 99 of file fb-doc_emitters.bi.

Here is the call graph for this function:

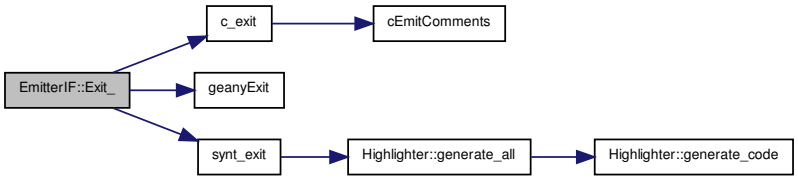


14.2.2.11 void EmitterIF::Exit_ (void) [inline]

function called after parsing the source code

Definition at line 102 of file fb-doc_emitters.bi.

Here is the call graph for this function:



14.2.2.12 void EmitterIF::CTOR_ (void) [inline]

function called at program start-up (once)

Definition at line 105 of file fb-doc_emitters.bi.

14.2.2.13 void EmitterIF::DTOR_ (void) [inline]

function called at program end (once)

Definition at line 108 of file fb-doc_emitters.bi.

14.2.3 Member Data Documentation

14.2.3.1 STRING EmitterIF::Nam = ""

the emitters name

Definition at line 52 of file fb-doc_emitters.bi.

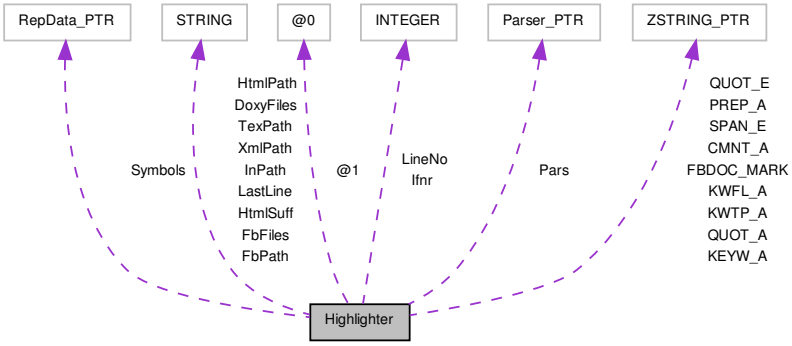
The documentation for this class was generated from the following file:

- src/fb-doc_emitters.bi

14.3 Highlighter Class Reference

Class to process the syntax highlighting.

Collaboration diagram for Highlighter:



Public Types

- enum WordTypes {
  FB_CODE, FB_KEYW, FB_KWTP, FB_KWFL,
  FB_PREP, FB_SYMB }

  The high-lighting categories.

Public Member Functions

- Highlighter ()
- Highlighter (BYVAL_AS_Parser_PTR)

  Constructor executing the complete process.
- SUB doDoxy (BYREF_AS_STRING)

  Constructor executing the complete process.
- SUB do_files ()

  Operate on all files.
- STATIC FUNCTION_AS_STRING prepare_tex (BYVAL_AS_Highlighter_PTR)

  Prepare a LaTeX file for syntax repairing.
- STATIC FUNCTION_AS_STRING prepare_xml (BYVAL_AS_Highlighter_PTR)

  Prepare a XML file for syntax repairing.
- STATIC FUNCTION_AS_STRING prepare_html (BYVAL_AS_Highlighter_PTR)

  Prepare a HTML file for syntax repairing.
- SUB generate_all (BYVAL_AS_ZSTRING_PTR, BYVAL_AS_INTEGER)

  Emit source code with syntax highlighting.
- FUNCTION_AS_STRING generate_code (BYVAL_AS_ZSTRING_PTR, BYVAL_AS_INTEGER, BYVAL_A-S_INTEGER)

  highlight normal source code

- FUNCTION_AS_ZSTRING_PTR word_type (BYREF_AS_STRING)

  Check type of word.
- FUNCTION_AS_STRING eol (BYVAL_AS_RepData_PTR, BYref_AS_INTEGER)

  the function called to end a line and start a new one
- FUNCTION_AS_STRING prepare (BYVAL_AS_Highlighter_PTR)

  the function called to extract links from original files
- FUNCTION_AS_STRING special_chars (BYVAL_AS_UBYTE_PTR, BYVAL_AS_INTEGER, BYVAL_AS_INTEGER)

  the function called for normal code to replace special characters

Public Attributes

- STRING FbPath

  The path to read FB code files from.
- STRING FbFiles

  A list of all FB file names.
- STRING InPath

  The path to read Doxygen files from.
- STRING DoxyFiles

  A list of all Doxygen file names.
- STRING HtmlPath

  The path for html files.
- STRING HtmlSuff

  The filename suffix for html files.
- STRING TexPath

  The path for LaTeX files.
- STRING XmlPath

  The path for XML files.
- STRING LastLine

  The last line red from the input file.
- RepData_PTR Symbols

  The list of linked symbols.
- Parser_PTR Pars

  The parser to operate with.
- ZSTRING_PTR FBDOC_MARK = @"@"<!-- Syntax-highlighting by fb-doc -->"

  Text to mark the output.
- ZSTRING_PTR KEYW_A = @"@""

  Code to start highlighting a keyword.
- ZSTRING_PTR KWTP_A = @"@""

  Code to start highlighting a keywordtype.
- ZSTRING_PTR KWFL_A = @"@""

  Code to start highlighting a flow keyword (not used yet)
- ZSTRING_PTR PREP_A = @"@""

  Code to start highlighting a preprocessor statement.
- ZSTRING_PTR CMNT_A = @"@""

  Code to start highlighting a comment.
- ZSTRING_PTR SPAN_E = @"@""

  Code to end highlighting.
- ZSTRING_PTR QUOT_A = @"@"""

  Code to start highlighting a string literal.

- ZSTRING_PTR QUOT_E = @"@"""

  Code to end highlighting a string literal.
- INTEGER Ifnr

  The file number for input.
- INTEGER LineNo

  The current line number.
- union {
  class {
  INTEGER **GenHtml**: 1
  Flag for html output.
  INTEGER **GenTex**: 1
  Flag for LaTeX output.
  INTEGER **GenXml**: 1
  Flag for XML output.
  }
  INTEGER **GenAny**
  All output flags.
  };

14.3.1 Detailed Description

Class to process the syntax highlighting.

The class is used to process the replacement of the Doxygen syntax highlighting for HTML, LaTeX and XML output. It contains members to

- scan for the files (Doxygen outputs),
- to read files and extract links,
- copy context to new files (header and footer)
- replace original file by fixed version.

Definition at line 254 of file fb-doc_emit_syntax.bas.

14.3.2 Member Enumeration Documentation

14.3.2.1 enum Highlighter::WordTypes

The high-lighting categories.

Enumerator

FB_CODE Normal code, no high-lighting.

FB_KEYW A keyword.

FB_KWTP A keyword type.

FB_KWFL A flow keyword (currently not used)

FB_PREP A preprocessor statement.

FB_SYMB A linked Symbol.

Definition at line 256 of file fb-doc_emit_syntax.bas.

14.3.3 Constructor & Destructor Documentation

14.3.3.1 Highlighter::Highlighter ( )

14.3.3.2 Highlighter::Highlighter (BYVAL_AS_Parser_PTR *P*)

Constructor executing the complete process.

Parameters

P	The parser for input

This constructor connects to the parser in use.

Definition at line 329 of file fb-doc_emit_syntax.bas.

14.3.4 Member Function Documentation

14.3.4.1 SUB Highlighter::doDoxy (BYREF_AS_STRING *Fnam*)

Constructor executing the complete process.

Parameters

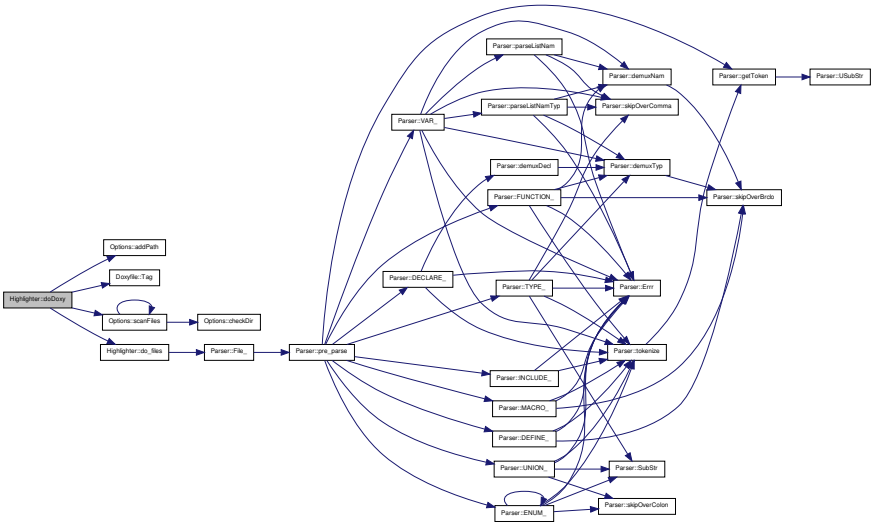
Fnam	The Doxyfile for input

This SUB controls the complete repairing process

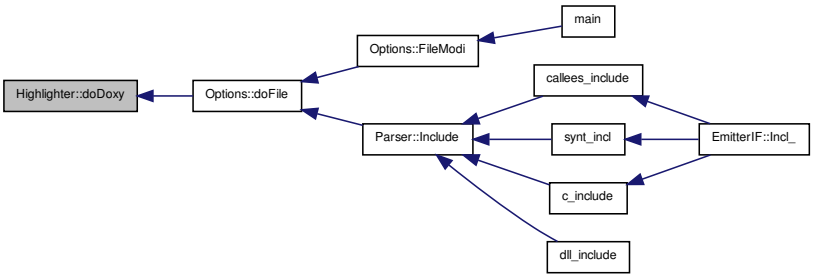
- load and parse Doxyfile for paths and output types
- scan FB directory for file names
- switch to matching emitter parameters
- scan output directories and process files

Definition at line 346 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.3.4.2 SUB Highlighter::do_files ( )

Operate on all files.

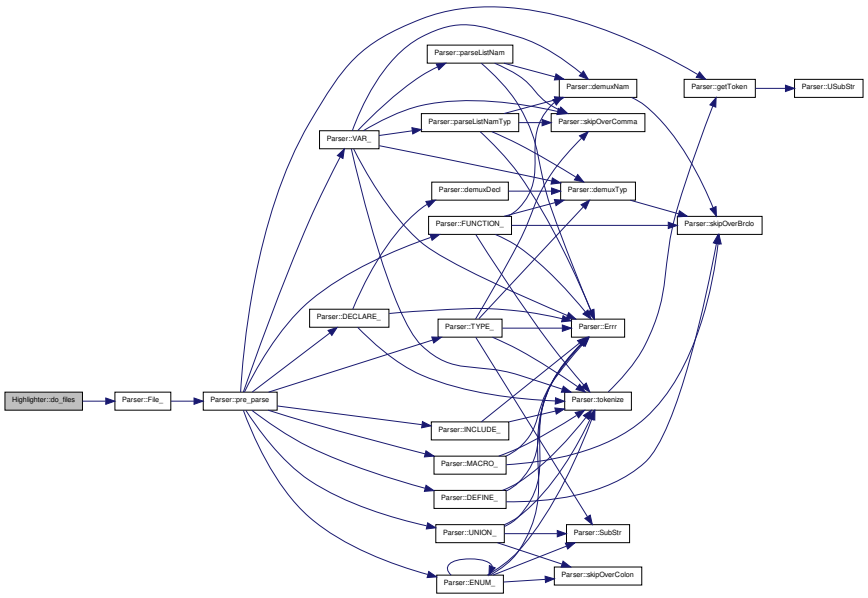
The procedure operates on all file names specified in DoxyFiles. It opens the input file, generated by Doxygen, writes a new file to copy the original context to (file start and end) and starts the emitter to replaces the source code section with advanced syntax highlighting.

Each file can only be fixed once (because the link texts from the original files are used and they change during the operation).

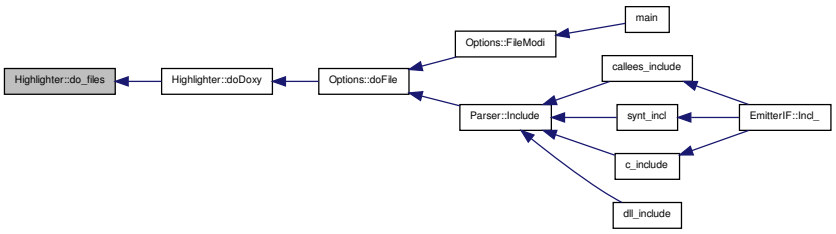
When done, the original file is killed and replaced by the new file, which is renamed to the original file name.

Definition at line 476 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.3.4.3 FUNCTION_AS_STRING Highlighter::prepare_tex (BYVAL_AS_Highlighter_PTR *Hgh*)

Prepare a LaTeX file for syntax repairing.

Parameters

Hgh	The Highlighter to operate with

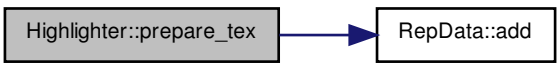
Returns

The file name of the FB source code

This function prepares a LaTeX file to replace the syntax highlighting. It reads the header from the original output and copies the context to the replacement file. The file name of the odiginal FB source is extracted from the header. Then the links from the original listing part are extracted in to the Highlighter::Symbols table.

Definition at line 1341 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



14.3.4.4 FUNCTION_AS_STRING Highlighter::prepare_xml (BYVAL_AS_Highlighter_PTR *Hgh*)

Prepare a XML file for syntax repairing.

Parameters

Hgh	The Highlighter to operate with

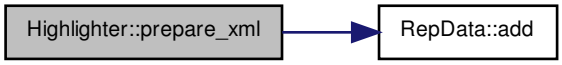
Returns

The file name of the FB source code

This function prepares a HTML file to replace the syntax highlighting. It reads the header from the original output and copies the context to the replacement file. The file name of the odiginal FB source is extracted from the header. Then the links from the original listing part are extracted in to the Highlighter::Symbols table.

Definition at line 1405 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



14.3.4.5 FUNCTION_AS_STRING Highlighter::prepare_html (BYVAL_AS_Highlighter_PTR *Hgh*)

Prepare a HTML file for syntax repairing.

Parameters

Hgh	The Highlighter to operate with

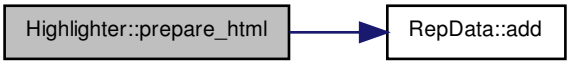
Returns

The file name of the FB source code

This function prepares a HTML file to replace the syntax highlighting. It reads the header from the original output and copies the context to the replacement file. The file name of the odiginal FB source is extracted from the header. Then the links from the original listing part are extracted in to the Highlighter::Symbols table.

Definition at line 1282 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



14.3.4.6 SUB Highlighter::generate_all (BYVAL_AS_ZSTRING_PTR *Buf,* BYVAL_AS_INTEGER *Stop_*)

Emit source code with syntax highlighting.

Parameters

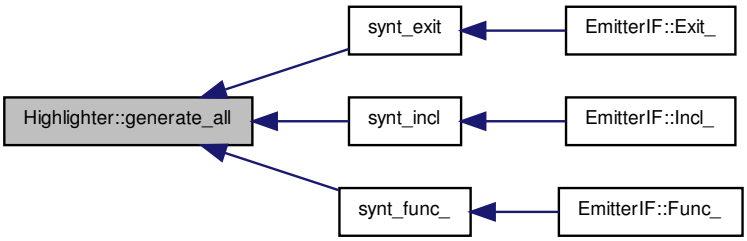
Buf	The buffer to read from
*Stop_*	The position to stop at

This procedure parses a source code section, starting at Parser::SrcBgn up to the given parameter Stop_. It operates on all kind of code (strings, comments and normal code). The output gets written to the Options::Ocha file.

Definition at line 534 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:

Here is the caller graph for this function:



14.3.4.7 FUNCTION_AS_STRING Highlighter::generate_code (BYVAL_AS_ZSTRING_PTR *T*, BYVAL_AS_INTEGER *A*, BYVAL_AS_INTEGER *L*)

highlight normal source code

Parameters

T	The input buffer from the parser
A	The start of the part to operate on (zero based)
L	The length of the part to operate on

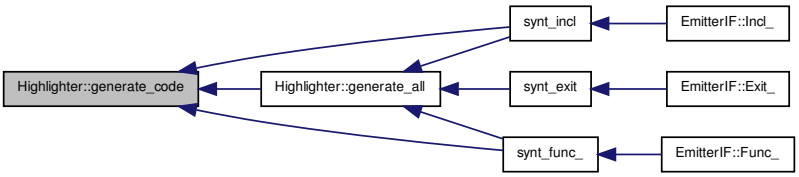
Returns

Formated html code for the input context

This function highlights a piece of code. It doesn't handle comments nor string literals. In normal code it separates keywords, preprocessors and symbols and encovers the context by the matching tags. In the the result special characters get replaced, like '&' by '&', '<' by '<', ... (special characters are different in HTML, LaTeX and XML output).

Definition at line 1211 of file fb-doc_emit_syntax.bas.

Here is the caller graph for this function:



14.3.4.8 FUNCTION_AS_ZSTRING_PTR Highlighter::word_type (BYREF_AS_STRING *W*)

Check type of word.

Parameters

W	The word to check

Returns

The type of the word (first byte) & formated word

Check a word in the source context and return its type and a string in camel-case letters for formating in mixed cases. The first byte of this string is the word type (1 = keyword, 2 = preprocessor).

Definition at line 599 of file fb-doc_emit_syntax.bas.

14.3.4.9 FUNCTION_AS_STRING Highlighter::eol (BYVAL_AS_RepData_PTR , BYref_AS_INTEGER)

the function called to end a line and start a new one

14.3.4.10 FUNCTION_AS_STRING Highlighter::prepare (BYVAL_AS_Highlighter_PTR)

the function called to extract links from original files

14.3.4.11 FUNCTION_AS_STRING Highlighter::special_chars (BYVAL_AS_UBYTE_PTR , BYVAL_AS_INTEGER , BYVAL_AS_INTEGER)

the function called for normal code to replace special characters

14.3.5 Member Data Documentation

14.3.5.1 STRING Highlighter::FbPath

The path to read FB code files from.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.2 STRING Highlighter::FbFiles

A list of all FB file names.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.3 STRING Highlighter::InPath

The path to read Doxygen files from.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.4 STRING Highlighter::DoxyFiles

A list of all Doxygen file names.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.5 STRING Highlighter::HtmlPath

The path for html files.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.6 STRING Highlighter::HtmlSuff

The filename suffix for html files.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.7 STRING Highlighter::TexPath

The path for LaTeX files.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.8 STRING Highlighter::XmlPath

The path for XML files.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.9 STRING Highlighter::LastLine

The last line red from the input file.

Definition at line 266 of file fb-doc_emit_syntax.bas.

14.3.5.10 RepData_PTR Highlighter::Symbols

The list of linked symbols.

Definition at line 275 of file fb-doc_emit_syntax.bas.

14.3.5.11 Parser_PTR Highlighter::Pars

The parser to operate with.

Definition at line 276 of file fb-doc_emit_syntax.bas.

14.3.5.12 ZSTRING_PTR Highlighter::FBDOC_MARK = @"@"<!-- Syntax-highlighting by fb-doc -->"

Text to mark the output.

Definition at line 278 of file fb-doc_emit_syntax.bas.

14.3.5.13 ZSTRING_PTR Highlighter::KEYW_A = @"@""

Code to start highlighting a keyword.

Definition at line 279 of file fb-doc_emit_syntax.bas.

14.3.5.14 ZSTRING_PTR Highlighter::KWTP_A = @"@""

Code to start highlighting a keywordtype.

Definition at line 280 of file fb-doc_emit_syntax.bas.

14.3.5.15 ZSTRING_PTR Highlighter::KWFL_A = @"@""

Code to start highlighting a flow keyword (not used yet)

Definition at line 281 of file fb-doc_emit_syntax.bas.

14.3.5.16 ZSTRING_PTR Highlighter::PREP_A = @"@""

Code to start highlighting a preprocessor statement.

Definition at line 282 of file fb-doc_emit_syntax.bas.

14.3.5.17 ZSTRING_PTR Highlighter::CMNT_A = @"@""

Code to start highlighting a comment.

Definition at line 283 of file fb-doc_emit_syntax.bas.

14.3.5.18 ZSTRING_PTR Highlighter::SPAN_E = @"@""

Code to end highlighting.

Definition at line 284 of file fb-doc_emit_syntax.bas.

14.3.5.19 ZSTRING_PTR Highlighter::QUOT_A = @"@"""

Code to start highlighting a string literal.

Definition at line 285 of file fb-doc_emit_syntax.bas.

14.3.5.20 ZSTRING_PTR Highlighter::QUOT_E = @"@"""

Code to end highlighting a string literal.

Definition at line 286 of file fb-doc_emit_syntax.bas.

14.3.5.21 INTEGER Highlighter::lfnr

The file number for input.

Definition at line 288 of file fb-doc_emit_syntax.bas.

14.3.5.22 INTEGER Highlighter::LineNo

The current line number.

Definition at line 288 of file fb-doc_emit_syntax.bas.

14.3.5.23 union { ... }

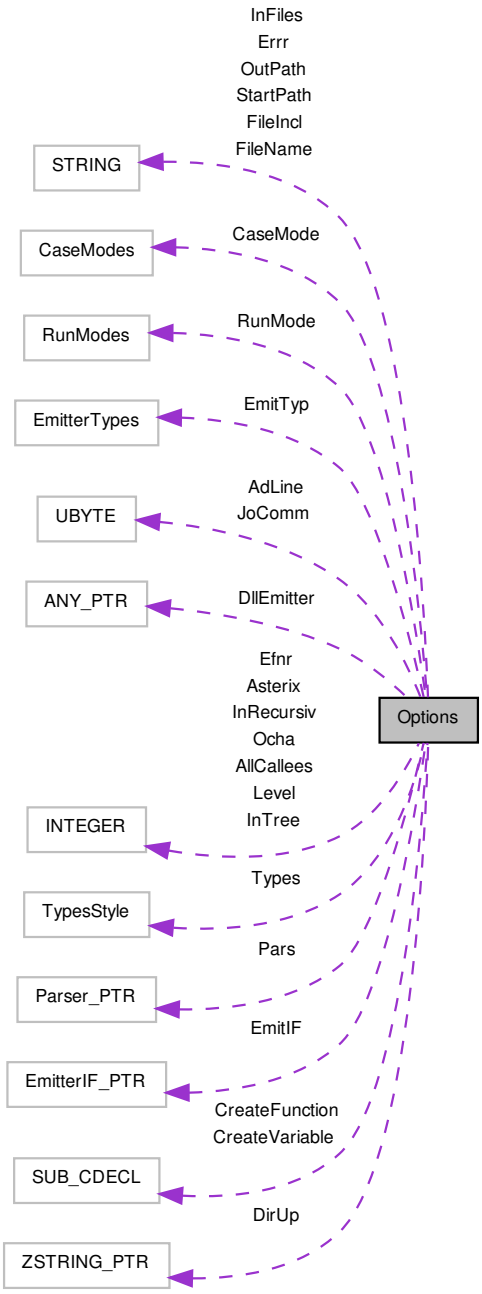
The documentation for this class was generated from the following file:

- src/fb-doc_emit_syntax.bas

14.4 Options Class Reference

Parameters red from the command line.

Collaboration diagram for Options:



Public Types

- enum RunModes {
  ERROR_MESSAGE, HELP_MESSAGE, VERSION_MESSAGE, GEANY_MODE,
  DEF_MODE, FILE_MODE, LIST_MODE, SYNT_MODE }

  fb-doc operation modes.
- enum EmitterTypes {

C_SOURCE, FUNCTION_NAMES, GTK_DOC_TEMPLATES, DOXYGEN_TEMPLATES, SYNTAX_REPAIR, EXTERNAL }

In-build emitters.

- enum TypesStyle { C_STYLE, FB_STYLE }

  The style of the types in the C source.
- enum CaseModes { CASE_ORIGN, CASE_LOWER, CASE_MIXED, CASE_UPPER }

  The letter-case mode for keywords.

Public Member Functions

- Options ()

  Read options and parameters from the command line.
- ~Options ()

  The destructor.
- FUNCTION_AS_RunModes parseCLI ()

  parse the command line
- FUNCTION_AS_string parseOptpara (byref_AS_INTEGER Idx)

  parse an additional parameter for an option
- SUB chooseEmitter (BYREF_AS_STRING F)

  Choose a named emitter, if possible.
- SUB FileModi ()

  Operate on file(s) and / or pattern(s)
- SUB doFile (BYREF_AS_STRING)

  Operate a single file.
- FUNCTION_AS_INTEGER checkDir (BYREF_AS_STRING)

  Create folder (if not exists)
- FUNCTION_AS_STRING scanFiles (BYREF_AS_STRING, BYREF_AS_STRING)

  Scan file names of given pattern.
- FUNCTION_AS_STRING addPath (BYREF_AS_STRING, BYREF_AS_STRING)

  Add two directories.

Public Attributes

- RunModes RunMode = DEF_MODE

  the mode to operate (defaults to DOXYFILTER)
- TypesStyle Types = FB_STYLE

  the style for the type generation (defaults to FB_STYLE)
- Parser_PTR Pars = 0

  the parser we use
- EmitterTypes EmitTyp = C_SOURCE

  the emitter type (defaults to C_Source)
- CaseModes CaseMode = CASE_ORIGN

  the emitter type (defaults to C_Source)
- EmitterIF_PTR EmitIF = 0

  the emitter we use (set in Options::parseCLI())
- ANY_PTR DllEmitter = 0

  the pointer for an external emitter
- ZSTRING_PTR DirUp = "../"
- STRING InFiles = ""

  name or pattern of all input file[s]

- STRING FileName = ""

  name and path of current input file
- STRING StartPath = ""

  path at program start
- STRING FileIncl = ""

  names of #INCLUDE files
- STRING OutPath = ""

  path for file output (option --outpath)
- STRING Errr = ""

  path for file output (option --outpath)
- INTEGER Asterix = 0

  style for C_Source emitter to export comment blocks
- INTEGER AllCallees = 0

  export external callee names as well (option --list-mode)
- INTEGER InRecursiv = 0

  flag set when InFiles should get scaned recursiv in subfolders
- INTEGER InTree = 0

  flag set when source tree should get scaned
- INTEGER Level = 0

  counter for #INCLUDEs
- INTEGER Efnr = 0

  file number for error messages (file modes)
- INTEGER Ocha = 0

  file number for output
- UBYTE JoComm = ASC("*")

  magic character to start a documentation comment
- UBYTE AdLine = ASC("&")

  magic character to start a comment for direct export
- EmitFunc CreateFunction

  emitter for a function declaration with parameter list
- EmitFunc CreateVariable

  emitter for a variable declaration

14.4.1 Detailed Description

Parameters red from the command line.

The class to scan options and parameters from command line. the command line gets parsed once at the program start. Since this structure is global, the settings are available in all internal modules.

Definition at line 34 of file fb-doc_options.bi.

14.4.2 Member Enumeration Documentation

14.4.2.1 enum Options::RunModes

fb-doc operation modes.

Enumerator

ERROR_MESSAGE Report an error found in the command line. The parser found an error in the command line and we cannot operate. Output the error message and stop.

HELP_MESSAGE Print the helptext. Stop after printing out some help information on how to start fb-doc and how to use options at the command line. (Option `--help`).

VERSION_MESSAGE Print version information. Print out the version information and stop (Option `--version`).

GEANY_MODE Operate in Geany mode. Input from STDIN and output on STDOUT. Usually this is to generate templates in Geany, but it also can be used with pipes. (Option *–geany-mode*).

DEF_MODE Operate as Doxygen filter. Read a file and generate C-source to STDOUT. (Default mode).

FILE_MODE Operate in scan mode. Read input from one or more files, write output to files. Usually this is to generate pseudo C source output, but an alternative emitter can be specified. When no file name or pattern is specified, all ∗.bas and ∗.bi files in the current folder gets parsed. (Option `--file-mode`).

LIST_MODE Operate in list mode. Read input from one or more files, write output to a single file *fb-doc.lfn*. Generate a list of callee names in this file. When no file name or pattern is specified, all ∗*.bas* and ∗*.bi* files in the current folder gets parsed. (Option `--list-mode`).

SYNT_MODE Operate in syntax-highlighting mode. Read input from one or more files, write output to a several files. As input fb-doc reads files created by Doxygen, containing the source listings in the intermediate format. The file types depend on the settings in the Doxyfile. It may be ∗*.html,* ∗*.tex* and ∗*.xml*, depending on GENERATE_HTML & SOURCE_BROWSER, GENERATE_LATEX & LATEX_SOURCE_CODE and GENERATE_XML & XML_PROGRAMLISTING.

Definition at line 36 of file fb-doc_options.bi.

14.4.2.2 enum Options::EmitterTypes

In-build emitters.

By default these four emitters are available in fb-doc. The enumerators are used for default settings in the Options class. The user can choose the emitter by option `--emitter`. The parameter gets checked against the EmitterIF::Nam string (or parts of it).

Enumerator

C_SOURCE emit pseudo C source (default and option `--file-mode`)

FUNCTION_NAMES emit a list of function names (option `--list-mode`)

GTK_DOC_TEMPLATES emit templates for gtk-doc (option `--geany-mode gtk`)

DOXYGEN_TEMPLATES emit templates for Doxygen (option `--geany-mode doxy`)

SYNTAX_REPAIR fix syntax highlighting of Doxygen listings (option `--syntax-mode`)

EXTERNAL external emitter loaded as plugin

Definition at line 103 of file fb-doc_options.bi.

14.4.2.3 enum Options::TypesStyle

The style of the types in the C source.

A FB type can be translated to a C type or it can be shown as a pseudo type. Ie instead of the C type void the pseudo type SUB can be used.

Enumerator

C_STYLE types gets translated to C

FB_STYLE pseudo FB types are used

Definition at line 117 of file fb-doc_options.bi.

14.4.2.4 enum Options::CaseModes

The letter-case mode for keywords.

Enumerator

CASE_ORIGN Output original formating.

CASE_LOWER Output keywords in lower case.

CASE_MIXED Output keywords in mixed case.

CASE_UPPER Output keywords in upper case.

Definition at line 123 of file fb-doc_options.bi.

14.4.3 Constructor & Destructor Documentation

14.4.3.1 Options::Options ( )

Read options and parameters from the command line.

The constructor scans all command line arguments and checks for options and their parameters (, see Options for details). Options may be specified in short form (starting with a single minus character) or in human readable long form (starting with two minus characters). Some options may have an additinal parameter.

Each command line argument that is neither an option nor its parameter gets interpreted as a file name or pattern. fb-doc collects them in a queue and operates on this queue afterwards. The queue may have mixed entries (names and patterns). It's recommended to specify queue entries in single or double quotes.

Definition at line 27 of file fb-doc_options.bas.

14.4.3.2 Options::~Options ( )

The destructor.

Delete the memory used for an external emitter (if any) and for the Parser.

Definition at line 42 of file fb-doc_options.bas.

14.4.4 Member Function Documentation

14.4.4.1 FUNCTION_AS_RunModes Options::parseCLI ( )

parse the command line

Returns

the RunMode

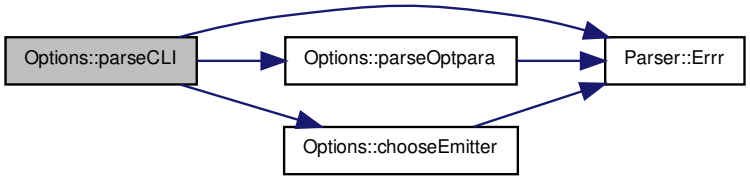
This function parses the command line. Options and its parameter (if required) get read and the file specifier(s) are listred in the variable InFiles, separated by new line characters.

The function returns the value of RunMode. In case of an error the RunMode is ERROR_MESSAGE and the variable Errr contains an message text. fb-doc stops execution in that case.

Loading an external emitter plugin must be done after scanning the COMMAND strings because DYLIBLOAD destroys the COMMAND array.

Definition at line 87 of file fb-doc_options.bas.

Here is the call graph for this function:



14.4.4.2 FUNCTION_AS_STRING Options::parseOptpara (byref_AS_INTEGER *idx*)

parse an additional parameter for an option

Parameters

idx	the current index in `COMMAND()`

Returns

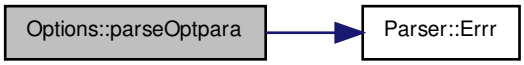
the parameter, without quotes if any

This function evaluates a parameter for an option. Some options need an additional parameter (ie like `--outpath`). It gets read by this function, removing surrounding single or double quotes (if any).

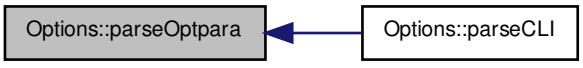
In case of no further parameter or when the parameter starts by an "'-'" character an error messages gets created.

Definition at line 61 of file fb-doc_options.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.4.4.3 SUB Options::chooseEmitter (BYREF_AS_STRING *F*)

Choose a named emitter, if possible.

Parameters

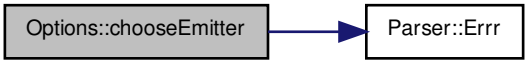
F	the name of the emitter to search for

This function checks for an emitter specified by the parameter F. The check is not case-sensitve and is done for a complete emitter name as well as for a fragment. So it's enough to specify some of the start characters of the emitter name (ie *dox* instead of *DoxygenTemplates*).

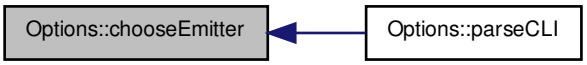
In case of no match in any internal EmitterIF::Nam this SUB tries to load an external emitter. If this fails an error message gets created and fb-doc stops execution.

Definition at line 165 of file fb-doc_options.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



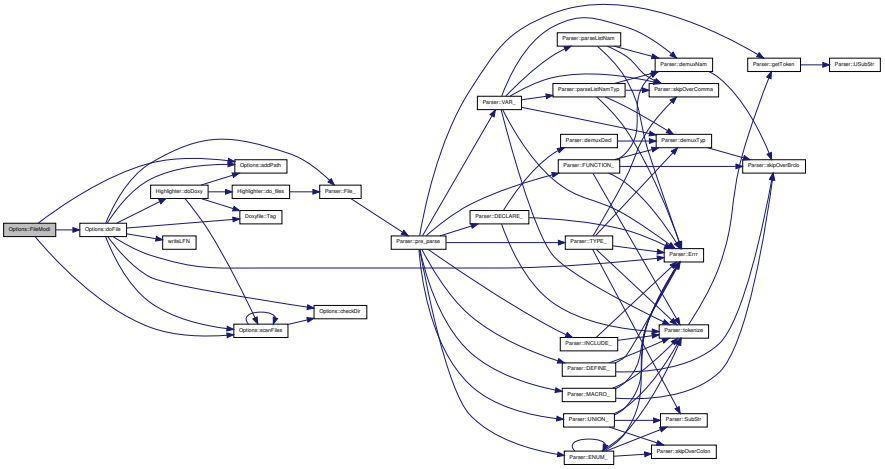
14.4.4.4 SUB Options::FileModi ( )

Operate on file(s) and / or pattern(s)

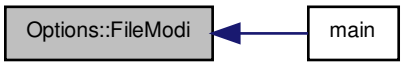
This SUB gets called in case of file input modes. It separates the file name specifiers from the Options::InFiles list, expands file patterns and executes the required operations.

Definition at line 297 of file fb-doc_options.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



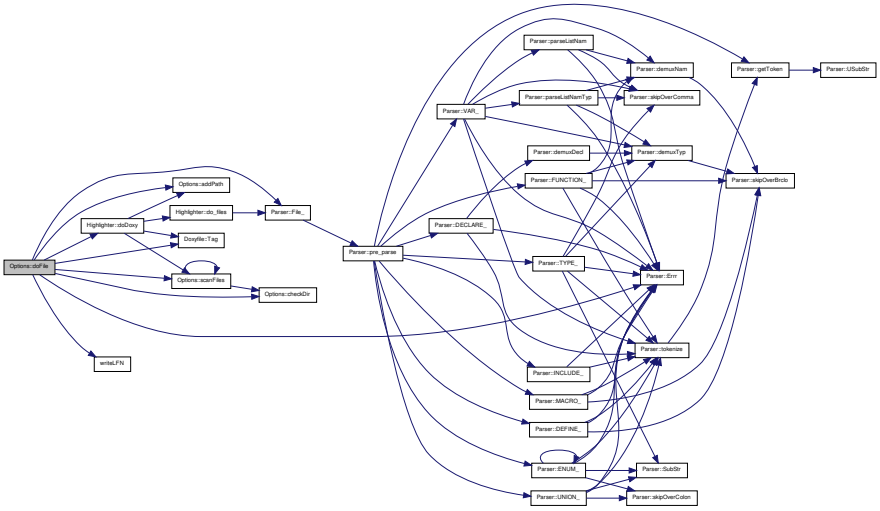
14.4.4.5 SUB Options::doFile (BYREF_AS_STRING *Fnam*)

Operate a single file.

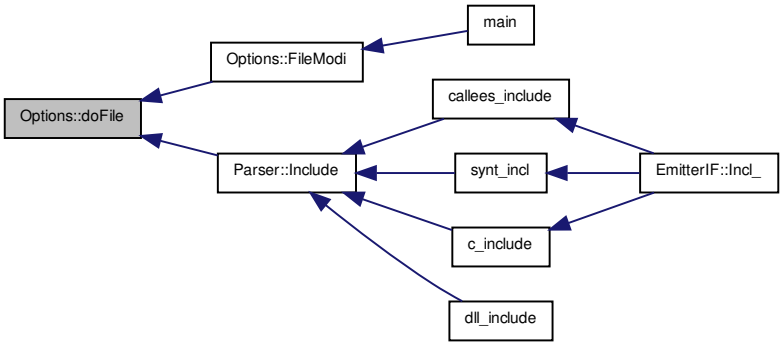
This SUB gets called to operate on a single file. Depending on the RunMode it emits messages to STDERR and it opens a file for the output channel Ocha.

Definition at line 384 of file fb-doc_options.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.4.4.6 FUNCTION_AS_INTEGER Options::checkDir (BYREF_AS_STRING *P*)

Create folder (if not exists)

Parameters

P	the folder name

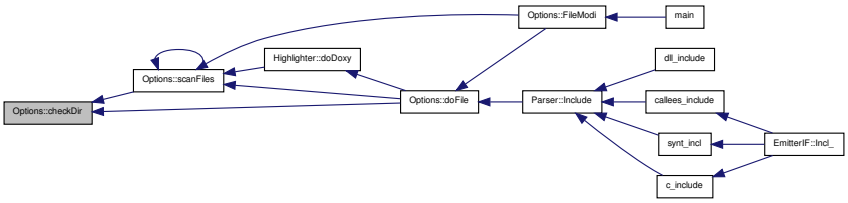
Returns

-1 on error, else 0

This FUNCTION gets called to prepare the folders for file output in `--file-mode`. It creates matching subfolders in the target directory set by option `--outpath` or its default.

Definition at line 271 of file fb-doc_options.bas.

Here is the caller graph for this function:



14.4.4.7 FUNCTION_AS_STRING Options::scanFiles (BYREF_AS_STRING *Patt*, BYREF_AS_STRING *Path*)

Scan file names of given pattern.

Parameters

Patt	The name pattern to search for
Path	A path to add

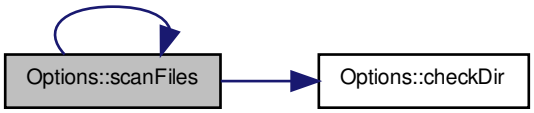
Returns

A list of file names and their subfolder

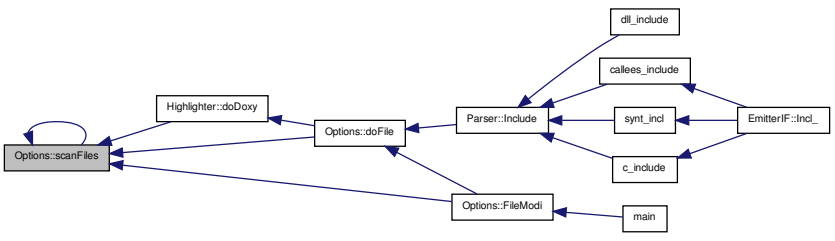
This function scans the current directory for file names matching a given pattern and the subfolders, if Options::InRecursiv ist set. It returns a list of filenames including their subfolder. The list is separated by newline characters.

Definition at line 200 of file fb-doc_options.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.4.4.8 FUNCTION_AS_STRING Options::addPath (BYREF_AS_STRING *P1*, BYREF_AS_STRING *P2*)

Add two directories.

Parameters

P1	the path of the basic directory
P2	the path of the directory to add

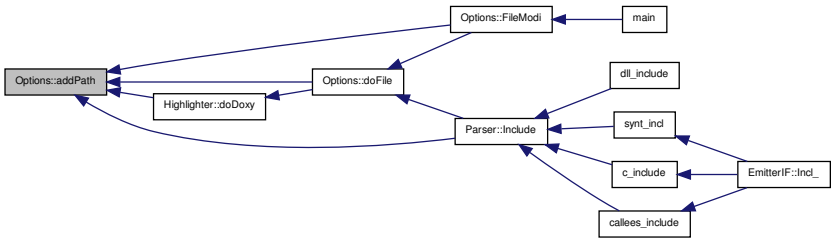
Returns

a string of the combined directory

Append second directory to first, if it's not an absolute path. In case of an absolute path this path gets returned. The returned path has a SLASH at the end and all '..' sequences are removed.

Definition at line 243 of file fb-doc_options.bas.

Here is the caller graph for this function:



14.4.5 Member Data Documentation

14.4.5.1 RunModes Options::RunMode = DEF_MODE

the mode to operate (defaults to DOXYFILTER)

Definition at line 130 of file fb-doc_options.bi.

14.4.5.2 TypesStyle Options::Types = FB_STYLE

the style for the type generation (defaults to FB_STYLE)

Definition at line 131 of file fb-doc_options.bi.

14.4.5.3 Parser_PTR Options::Pars = 0

the parser we use

Definition at line 132 of file fb-doc_options.bi.

14.4.5.4 EmitterTypes Options::EmitTyp = C_SOURCE

the emitter type (defaults to *C_Source*)

Definition at line 133 of file fb-doc_options.bi.

14.4.5.5 CaseModes Options::CaseMode = CASE_ORIGN

the emitter type (defaults to *C_Source*)

Definition at line 134 of file fb-doc_options.bi.

14.4.5.6 EmitterIF_PTR Options::EmitIF = 0

the emitter we use (set in Options::parseCLI())

Definition at line 135 of file fb-doc_options.bi.

14.4.5.7 ANY_PTR Options::DllEmitter = 0

the pointer for an external emitter

Definition at line 136 of file fb-doc_options.bi.

14.4.5.8 ZSTRING_PTR Options::DirUp = "../"

Definition at line 137 of file fb-doc_options.bi.

14.4.5.9 STRING Options::InFiles = ""

name or pattern of all input file[s]

Definition at line 143 of file fb-doc_options.bi.

14.4.5.10 STRING Options::FileName = ""

name and path of current input file

Definition at line 144 of file fb-doc_options.bi.

14.4.5.11 STRING Options::StartPath = ""

path at program start

Definition at line 145 of file fb-doc_options.bi.

14.4.5.12 STRING Options::FileIncl = ""

names of #`INCLUDE` files

Definition at line 146 of file fb-doc_options.bi.

14.4.5.13 STRING Options::OutPath = ""

path for file output (option `--outpath`)

Definition at line 147 of file fb-doc_options.bi.

14.4.5.14 STRING Options::Errr = ""

path for file output (option `--outpath`)

Definition at line 148 of file fb-doc_options.bi.

14.4.5.15 INTEGER Options::Asterix = 0

style for C_Source emitter to export comment blocks

Definition at line 150 of file fb-doc_options.bi.

14.4.5.16 INTEGER Options::AllCallees = 0

export external callee names as well (option `--list-mode`)

Definition at line 151 of file fb-doc_options.bi.

14.4.5.17 INTEGER Options::InRecursiv = 0

flag set when InFiles should get scaned recursiv in subfolders

Definition at line 152 of file fb-doc_options.bi.

14.4.5.18 INTEGER Options::InTree = 0

flag set when source tree should get scaned

Definition at line 153 of file fb-doc_options.bi.

14.4.5.19 INTEGER Options::Level = 0

counter for #`INCLUDE`s

Definition at line 154 of file fb-doc_options.bi.

14.4.5.20 INTEGER Options::Efnr = 0

file number for error messages (file modes)

Definition at line 155 of file fb-doc_options.bi.

14.4.5.21 INTEGER Options::Ocha = 0

file number for output

Definition at line 156 of file fb-doc_options.bi.

14.4.5.22 UBYTE Options::JoComm = ASC("*")

magic character to start a documentation comment

Definition at line 158 of file fb-doc_options.bi.

14.4.5.23 UBYTE Options::AdLine = ASC("&")

magic character to start a comment for direct export

Definition at line 159 of file fb-doc_options.bi.

14.4.5.24 EmitFunc Options::CreateFunction

emitter for a function declaration with parameter list

Definition at line 161 of file fb-doc_options.bi.

14.4.5.25 EmitFunc Options::CreateVariable

emitter for a variable declaration

Definition at line 161 of file fb-doc_options.bi.

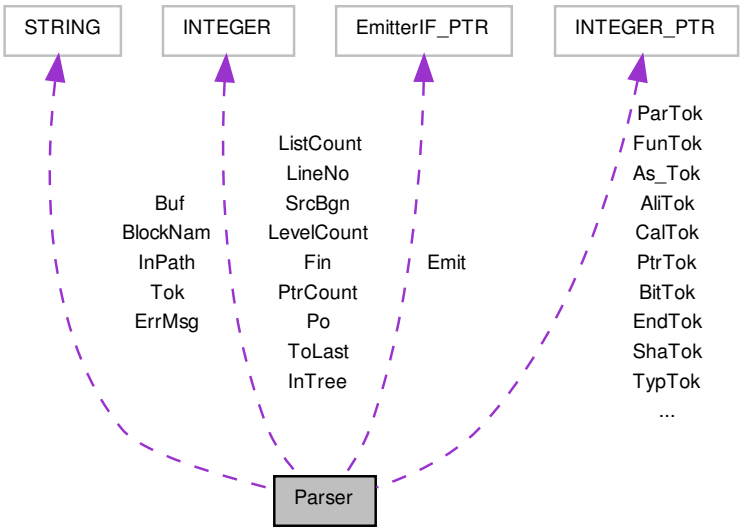
The documentation for this class was generated from the following files:

- src/fb-doc_options.bi
- src/fb-doc_options.bas

14.5 Parser Class Reference

The parser.

Collaboration diagram for Parser:



Public Types

- enum ParserTokens {
  MSG_STOP = -1, MSG_ERROR, TOK_EOS, TOK_BRCLO,
  TOK_COMMA, TOK_KLOPN, TOK_KLCLO, TOK_DOT,
  TOK_TRIDO, TOK_QUOTE, TOK_LATTE, TOK_BROPN,
  TOK_EQUAL, TOK_ABST, TOK_ALIA, TOK_AS,
  TOK_BYRE, TOK_BYVA, TOK_CDEC, TOK_CLAS,
  TOK_COMM, TOK_CONS, TOK_DECL, TOK_DEFI,
  TOK_DIM, TOK_END, TOK_ENUM, TOK_EXDS,
  TOK_EMAC, TOK_EXRN, TOK_EXPO, TOK_INCL,
  TOK_LIB, TOK_MACR, TOK_NAMS, TOK_ONCE,
  TOK_OVER, TOK_PASC, TOK_PRIV, TOK_PROT,
  TOK_RDIM, TOK_PTR, TOK_PUBL, TOK_SCOP,
  TOK_SHAR, TOK_STAT, TOK_STCL, TOK_TYPE,
  TOK_UNIO, TOK_VAR, TOK_VIRT, TOK_FUNC,
  TOK_FILD, TOK_OPER, TOK_PROP, TOK_SUB,
  TOK_CTOR, TOK_DTOR, TOK_BYTE, TOK_DOUB,
  TOK_INT, TOK_LONG, TOK_LINT, TOK_STRI,
  TOK_SHOR, TOK_SING, TOK_UBYT, TOK_ULNG,
  TOK_ULIN, TOK_UINT, TOK_USHO, TOK_WSTR,
  TOK_ZSTR, TOK_WORD, TOK_COMSL, TOK_COMML }

  The tokens used by the parser.

Public Member Functions

- Parser (BYVAL_AS_EmitterIF_PTR)

  The constructor.

Filehandlers

Filehandlers are used to load some FreeBasic source code input in to the buffer Parser::Buf from different input channels (may be the STDIN pipe, a single file or all files in a folder). Afterwards the source code gets parsed and translated output created by the emitter gets returned.

- SUB File_ (BYREF_AS_STRING, BYVAL_AS_INTEGER)

  Read a buffer from a file and parse.
- SUB StdIn ()

  Read a buffer from pipe STDIN and parse.
- SUB Include (BYVAL_AS_STRING)

  start a new parser to #INCLUDE a file
- STATIC SUB writeOut (BYREF_AS_STRING)

  write a piece of output (for external emitters only)

Properties to extract original source code

Properties are used to extract original text from the input buffer at a certain position. This is the current position of the parser, or the position of the given parameter.

- PROPERTY_AS_INTEGER_PTR CurTok ()

  The current token.
- PROPERTY_AS_STRING BitIni ()

  The initialization of a bitfield.
- PROPERTY_AS_STRING SubStr ()

  Context of current token.
- PROPERTY_AS_STRING SubStr (BYVAL_AS_INTEGER_PTR)

  Check current token position, extract word if string type.

Parsers for lists and blocks

External demuxers are called from emitter-handlers when they sit on a list of statments, ie in an ENUM block or in a DIM statement with more than one variable.

- SUB parseListNam (BYVAL_AS_EmitFunc)
  Evaluate a list of names.
- SUB parseListNamTyp (BYVAL_AS_EmitFunc)
  Evaluate a list of declarations.
- SUB parseListPara (BYVAL_AS_EmitFunc)
  Evaluate a parameter list.
- SUB parseBlockEnum (BYVAL_AS_EmitFunc)
  Evaluate the context of an ENUM block.
- SUB parseBlockTyUn (BYVAL_AS_EmitFunc)
  Evaluate the context of a block.

Public Attributes

Parser STRINGs

STRING variables to exchange data with th emitters.

- STRING InPath = ""
  path of current of input file (option `--tree`)
- STRING Buf
  the input buffer
- STRING ErrMsg
  an error message
- STRING Tok
  the token list (binaries)
- STRING BlockNam
  the name of a block (ENUM, UNION, TYPE)

Parser token pointers

Pointers to the token list, used to specify the result of the parsing process. The tokens specify the construct in progress and they are used by the emitters to create their output.

Four main tokens specify the kind of the construct:

- *StaTok: the start of the current construct*
- *NamTok - subtokens: DimTok, IniTok (may be zero in parameter lists)*
- *TypTok - subtokens: ShaTok, PtrTok, Co1Tok, Co2Tok (is zero for VAR, may be zero for CONST)*
- *FunTok - subtokens: ParTok, AliTok, DivTok, By_Tok (is zero for variables)*

Other tokens are only valid if the main token is not zero.

- INTEGER_PTR StaTok
  the pre-parsed token
- INTEGER_PTR NamTok
  the name token of the construct
- INTEGER_PTR DimTok
  the token of the left parenthesis of a dimension
- INTEGER_PTR IniTok
  the start token of an initializer ('=')
- INTEGER_PTR BitTok
  the start token of an bitfiled declaration (':')
- INTEGER_PTR TypTok
  the token of the type keyword
- INTEGER_PTR ShaTok
  the token of the SHARED keyword

- INTEGER_PTR PtrTok

  the token of the first PTR keyword
- INTEGER_PTR Co1Tok

  the token of the first CONST keyword (if any)
- INTEGER_PTR Co2Tok

  the token of the second CONST keyword (if any)
- INTEGER_PTR FunTok

  type of the pre-parsed token
- INTEGER_PTR CalTok

  the token of the calling convention keyword (if any)
- INTEGER_PTR AliTok

  the token of the ALIAS keyword (if any)
- INTEGER_PTR As_Tok

  the token of the AS keyword (if no SUB)
- INTEGER_PTR ParTok

  the token of the left parenthesis of a parameter list
- INTEGER_PTR By_Tok

  the token of the declaration specifier (BYVAL / BYREF)
- INTEGER_PTR DivTok

  the token of different purposes like the STATIC keyword in a member function declaration (if any) or the LIB keyword in a normal declaration
- INTEGER_PTR Tk1

  the first token in a statment
- INTEGER_PTR EndTok

  the last token in the list
- INTEGER_PTR UserTok

  a token for customised usage in emitter-handlers

Parser counters, integers and pointers

Diverse variables used on different events. They inform the emitters about the state of the parser.

- INTEGER Po

  the current position in the input buffer Parser::Buf
- INTEGER Fin

  the last position in the input buffer Parser::Buf
- INTEGER PtrCount

  the number of PTR keywords
- INTEGER SrcBgn

  start position to export FB source
- INTEGER LineNo

  the current line in the input buffer Parser::Buf
- INTEGER LevelCount

  the level in nested blocks (one based)
- INTEGER InTree

  flag to indicate if to follow source tree #INCLUDEs
- INTEGER ListCount

  the current entry in a list (zero based)

Private Types

- enum EoS_Modi { TO_END_BLOCK, TO_END, TO_EOL, TO_COLON }

  The modi for end of statement searching.

Private Member Functions

Internal parsers

Functions for evaluating a construct in the FreeBasic source code after a relevant keyword was found in the pre-parsering process. After the fine-parsing process the matching emitter-handler gets called. (Or the function EmitterIF::Error_() in case of a syntax problem. It's up to the emitter if and where the user sees the error message.)

- SUB pre_parse ()
  Pre-parse all FB source code, search relevant constructs.
- PROPERTY_AS_STRING USubStr ()
  Check current token position, extract word if string type.
- FUNCTION_AS_INTEGER getToken ()
  Check word at current parser position.
- FUNCTION_AS_INTEGER tokenize (BYVAL_AS_EoS_Modi)
  Parse a relevant construct, create a token list.
- FUNCTION_AS_INTEGER Errr (BYREF_AS_STRING)
  Emit an error message.
- FUNCTION_AS_INTEGER demuxNam (BYVAL_AS_INTEGER, BYVAL_AS_INTEGER)
  Evaluate a name, dimension and initializer in the token list.
- FUNCTION_AS_INTEGER demuxTyp (BYVAL_AS_INTEGER)
  Evaluate a type declaration in the token list.
- FUNCTION_AS_INTEGER demuxDecl ()
  Evaluate a function declaration in the token list.
- FUNCTION_AS_INTEGER skipOverColon ()
  Move to the end of statement and one step beyond.
- FUNCTION_AS_INTEGER skipOverBrclo ()
  Move to the matching right parenthesis and one step beyond.
- FUNCTION_AS_INTEGER skipOverComma ()
  Move to the next comma and one step beyond.
- FUNCTION_AS_INTEGER VAR_ ()
  Evaluate a variable declaration.
- FUNCTION_AS_INTEGER TYPE_ ()
  Evaluate a TYPE or CLASS statement.
- FUNCTION_AS_INTEGER ENUM_ ()
  Evaluate an ENUM block.
- FUNCTION_AS_INTEGER UNION_ ()
  Evaluate an UNION block.
- FUNCTION_AS_INTEGER FUNCTION_ ()
  Evaluate a function.
- FUNCTION_AS_INTEGER DECLARE_ ()
  Evaluate a forward declaration.
- FUNCTION_AS_INTEGER INCLUDE_ ()
  Evaluate an #INCLUDE line.
- FUNCTION_AS_INTEGER DEFINE_ ()
  Evaluate a #DEFINE declaration.
- FUNCTION_AS_INTEGER MACRO_ ()
  Evaluate a #MACRO declaration.

Private Attributes

Internal values

Variables used in pre-parsing process.

- INTEGER ToLast
  the type of the token before the current one
- INTEGER_PTR Tk
  the current token of the parser

- INTEGER_PTR A
  start of a word in pre-parsing
- INTEGER_PTR L
  length of a word in pre-parsing
- EmitterIF_PTR Emit
  the emitter interface

14.5.1 Detailed Description

The parser.

Class to handle FreeBasic source code. A Parser allways work on exactly one input stream, comming from a file or from STDIN. The Parser does

- read the source from an input channel (see StdIn(), File_())
- call function EmitterIF::Init_() before parsing
- pre-parse the source for relavant code fractions (see pre_parse()) and call the comments emitter-handler on the way.
- fine-scan any relavant code fractions (see tokenize()) and call the matching emitter-handler for this code (done by the private functions ie like DEFINE_(), FUNCTION_(), VAR_(), ...)
- provide functions to extract elements form the source (like SubStr(), CurTok(), BitIni(), ...)
- call matching emitter functions to generaate the output stream
- call function EmitterIF::Exit_() after parsing

When fb-doc follows the source tree (option `--tree`), the function EmitterIF::Incl_() creates a new Parser for each file.

Definition at line 50 of file fb-doc_parser.bi.

14.5.2 Member Enumeration Documentation

14.5.2.1 enum Parser::ParserTokens

The tokens used by the parser.

Enumerators used to classify the type of a token found in the FreeBasic source code.

Enumerator

MSG_STOP end of file / token list reached

MSG_ERROR create error message, continue parsing

TOK_EOS end of statement (either new line or `":"`)

TOK_BRCLO right parenthesis

TOK_COMMA a comma

TOK_KLOPN left other bracket

TOK_KLCLO right other bracket

TOK_DOT a dot

TOK_TRIDO three dots

TOK_QUOTE a string constant (including quotes)

TOK_LATTE the '#' character

TOK_BROPN left parenthesis

TOK_EQUAL the '=' character
TOK_ABST the ABSTRACT keyword
TOK_ALIA the ALIAS keyword
TOK_AS the AS keyword
TOK_BYRE the BYREF keyword
TOK_BYVA the BYVAL keyword
TOK_CDEC the CDECL keyword
TOK_CLAS the CLASS keyword
TOK_COMM the COMMON keyword
TOK_CONS the CONST keyword
TOK_DECL the DECLARE keyword
TOK_DEFI the #DEFINE keyword
TOK_DIM the DIM keyword
TOK_END the END keyword
TOK_ENUM the ENUM keyword
TOK_EXDS the EXTENDS keyword
TOK_EMAC the #ENDMACRO keyword
TOK_EXRN the EXTERN keyword
TOK_EXPO the EXPORT keyword
TOK_INCL the #INCLUDE keyword
TOK_LIB the LIB keyword
TOK_MACR the #MACRO keyword
TOK_NAMS the NAMESPACE keyword
TOK_ONCE the ONCE keyword
TOK_OVER the OVERLOAD keyword
TOK_PASC the PASCAL keyword
TOK_PRIV the PRIVATE keyword
TOK_PROT the PROTECTED keyword
TOK_RDIM the REDIM keyword
TOK_PTR the PTR or POINTER keyword
TOK_PUBL the PUBLIC keyword
TOK_SCOP the SCOPE keyword
TOK_SHAR the SHARED keyword
TOK_STAT the STAvarTIC keyword
TOK_STCL the STDCALL keyword
TOK_TYPE the TYPE keyword
TOK_UNIO the UNION keyword
TOK_VAR the VAR keyword
TOK_VIRT the VIRTUAL keyword
TOK_FUNC the FUNCTION keyword
TOK_FILD the FIELD keyword
TOK_OPER the OPERATOR keyword
TOK_PROP the PROPERTY keyword
TOK_SUB the SUB keyword
TOK_CTOR the CONSTRUCTOR keyword

TOK_DTOR the DESTRUCTOR keyword

TOK_BYTE the BYTE data type

TOK_DOUB the DOUBLE data type

TOK_INT the INTEGER data type

TOK_LONG the LONG data type

TOK_LINT the LONGINT data type

TOK_STRI the STRING data type

TOK_SHOR the SHORT data type

TOK_SING the SINGLE data type

TOK_UBYT the UBYTE data type

TOK_ULNG the ULONG data type

TOK_ULIN the ULONGINT data type

TOK_UINT the UINTEGER data type

TOK_USHO the USHORT data type

TOK_WSTR the ZSTRING data type

TOK_ZSTR the ZSTRING data type

TOK_WORD a word

TOK_COMSL line end comment (single line)

TOK_COMML multi line comment

Definition at line 57 of file fb-doc_parser.bi.

14.5.2.2 enum Parser::EoS_Modi [private]

The modi for end of statement searching.

Enumerators used in internal in the parser to specify where to stop to tokenize the input buffer

Enumerator

TO_END_BLOCK tokenize to the end of the structure

TO_END stop at the end of the token list

TO_EOL stop at the next line end

TO_COLON stop at the next line end or colon

Definition at line 274 of file fb-doc_parser.bi.

14.5.3 Constructor & Destructor Documentation

14.5.3.1 Parser::Parser (BYVAL_AS_EmitterIF_PTR *Em*)

The constructor.

Parameters

Em	a pointer to the emitter interface to use

Initialize the start values. We get the pointer to the EmitterIF to use and we create a short token list (just two entries) for usage inside the parser. This token list is static and it doesn't change its adress (unlike the list Parser::Tok that may shift when growing).

Definition at line 21 of file fb-doc_parser.bas.

14.5.4 Member Function Documentation

14.5.4.1 SUB Parser::File_ (BYREF_AS_STRING *File,* BYVAL_AS_INTEGER *Tree*)

Read a buffer from a file and parse.

Parameters

File	the name of the file to translate
Tree	if to follow source tree #INCLUDE

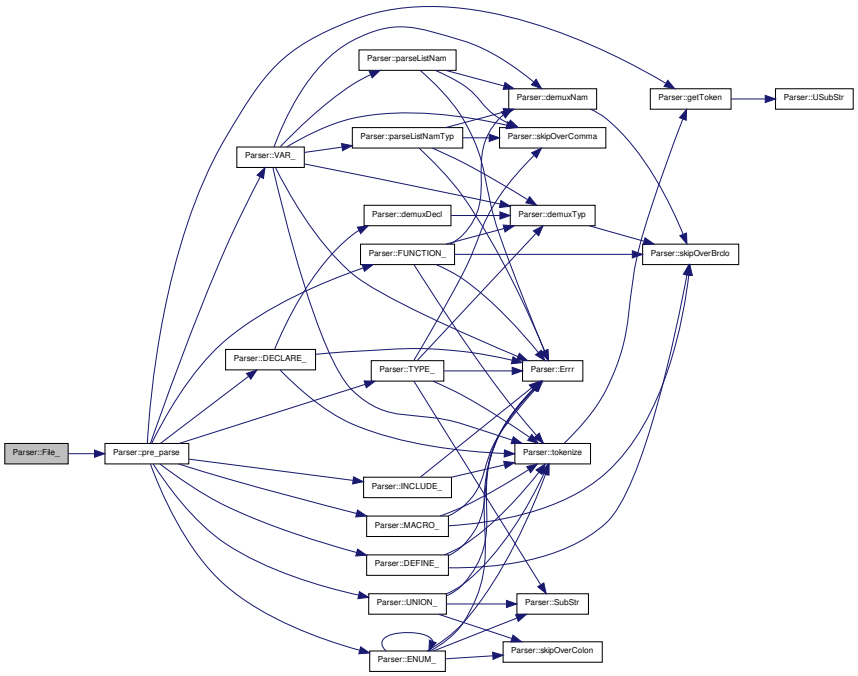
Returns

the translated code (if any)

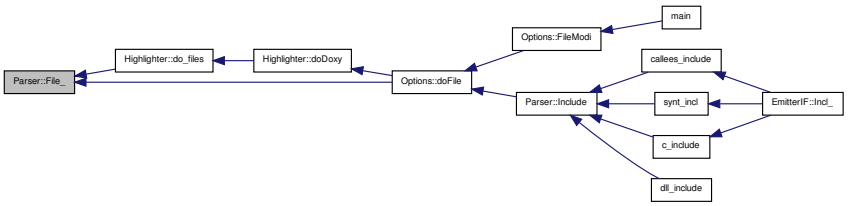
Read a file in to input buffer. Start detailed parsing on each relevant construct. Otherways skip the current line. Export comments on the way. Do nothing if file doesn't exist or isn't readable.

Definition at line 1093 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.2 SUB Parser::StdIn ( )

Read a buffer from pipe STDIN and parse.

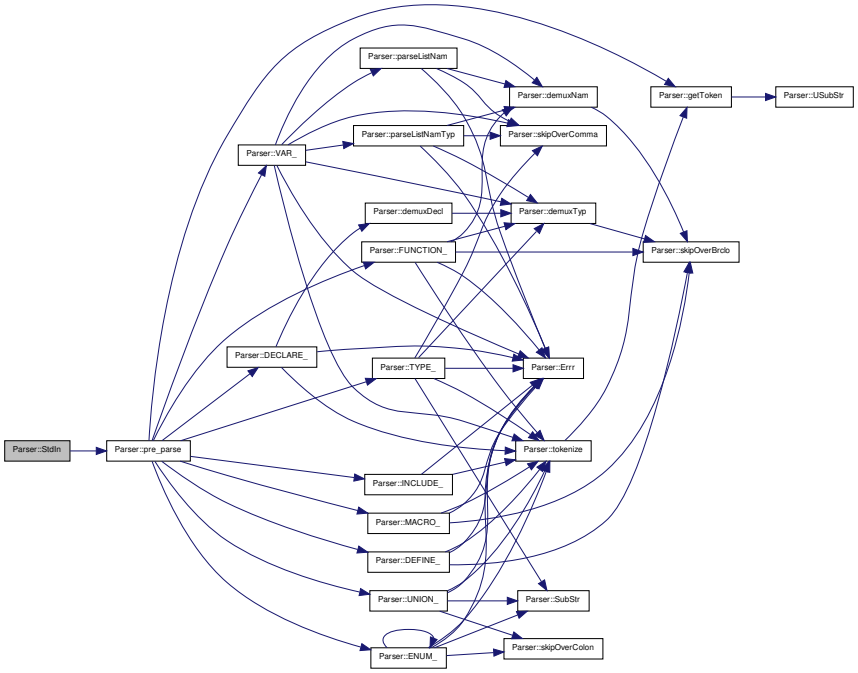
Returns

the translated code (or an informal text)

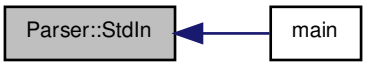
Get all characters from STDIN. Start the parsing process. If there is no input (an empty line) then call function EmitterIF::Empty_(). (Useful for generating file templates in mode `--geany-mode`.)

Definition at line 1118 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.3 SUB Parser::Include (BYVAL_AS_STRING *N*)

start a new parser to #`INCLUDE` a file

Parameters

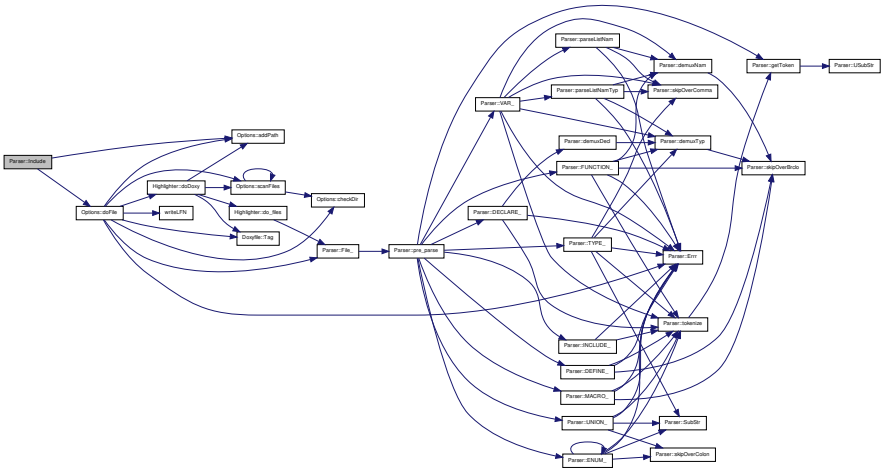
N	the (path, if any, and) file name

This procedure is used by the emitter handlers for #INCLUDE statements. It checks if the file hasn't been done already and can get opened. If one of these fails a message gets sent to STDERR and the file gets skipped.

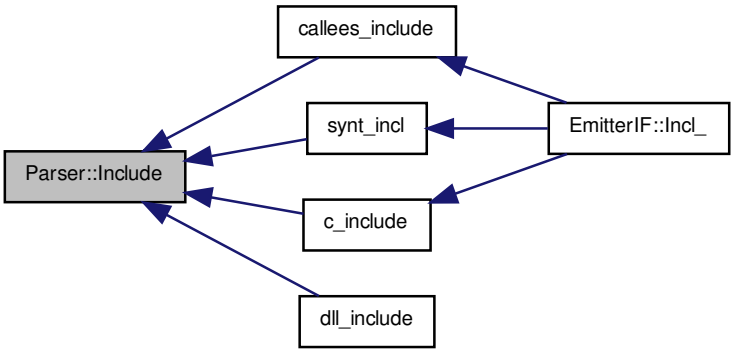
Otherwise a new parser gets started to operate on that file.

Definition at line 1293 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.4 SUB Parser::writeOut (BYREF_AS_STRING *T*)

write a piece of output (for external emitters only)

Parameters

T	the text to write

External emitters cannot use the streams opened in the main program directly. They have to send text to this procedure to use the standard output stream.

Definition at line 1155 of file fb-doc_parser.bas.

14.5.4.5 PROPERTY_AS_INTEGER_PTR Parser::CurTok ( )

The current token.

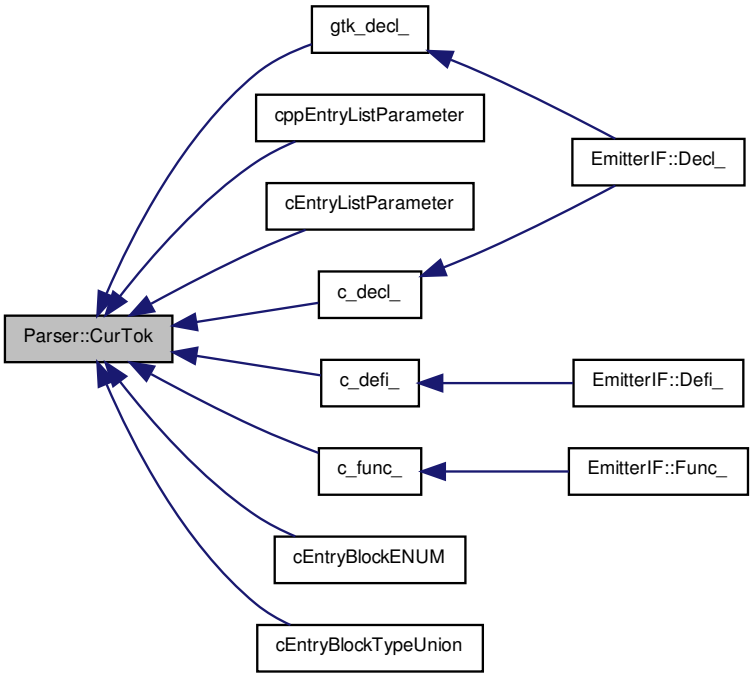
Returns

the token the parser currently stopping at

This property returns the parser token (where the parser currently stops).

Definition at line 1167 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.6 PROPERTY_AS_STRING Parser::BitIni ( )

The initialization of a bitfield.

Returns

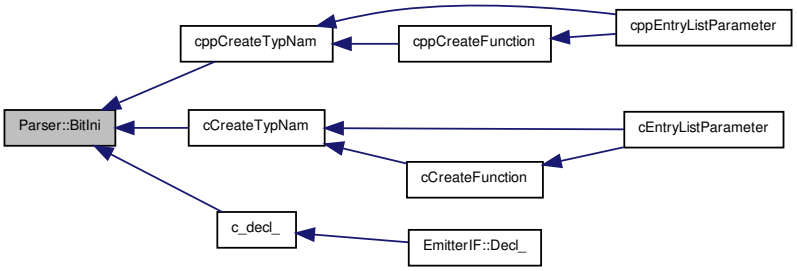
the bitfield initialization

This property returns the initialization of a bitfield in a TYPE / UNION block. The size of the bitfield may either be an integer number or a macro (= TOK_WORD).

Definition at line 1179 of file fb-doc_parser.bas.

Here is the call graph for this function:

Here is the caller graph for this function:



14.5.4.7 PROPERTY_AS_STRING Parser::SubStr ( )

Context of current token.

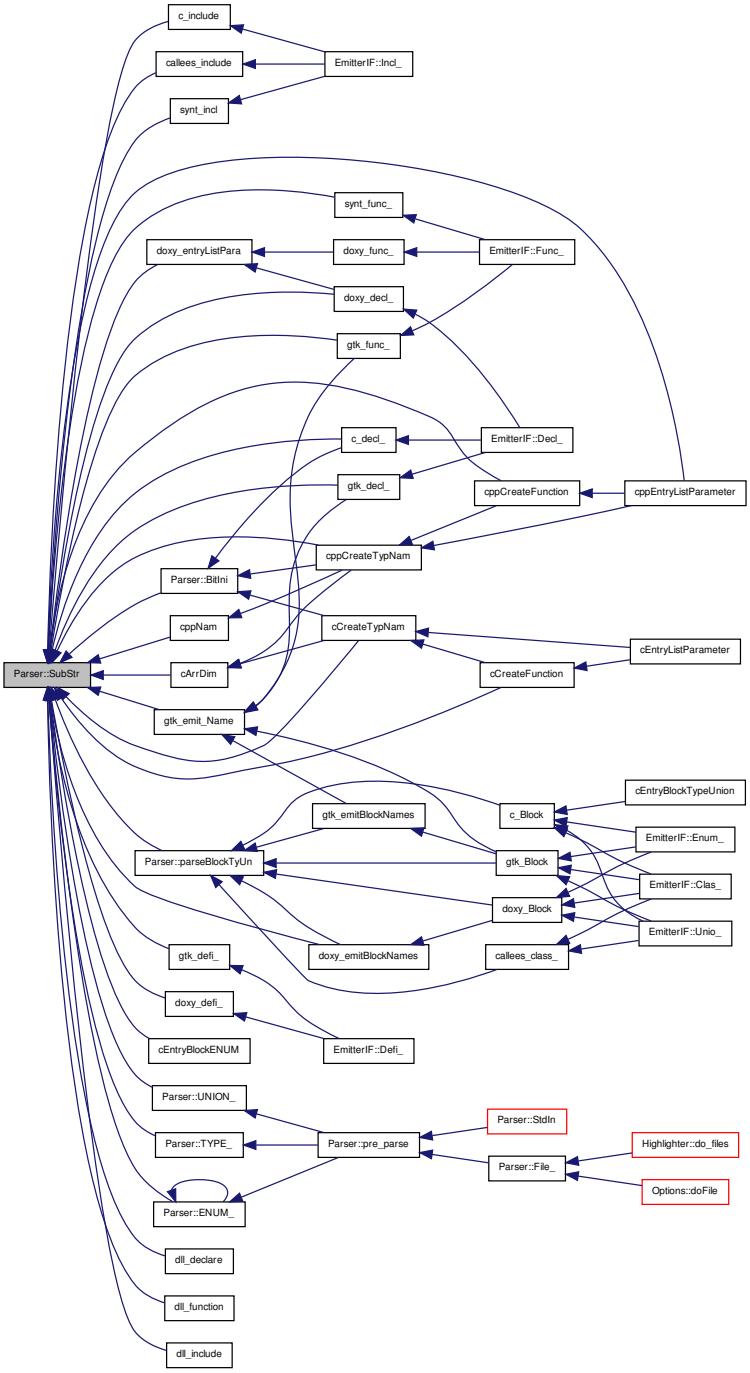
Returns

the context of the current token

This property returns the context of the current token from the input buffer.

Definition at line 1251 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.8 PROPERTY_AS_STRING Parser::SubStr (BYVAL_AS_INTEGER_PTR *T*)

Check current token position, extract word if string type.

Parameters

T	The token to read

Returns

the context of the token T

This property returns the context of the token T from the input buffer.

Definition at line 1264 of file fb-doc_parser.bas.

14.5.4.9 SUB Parser::parseListNam (BYVAL_AS_EmitFunc *Export_*)

Evaluate a list of names.

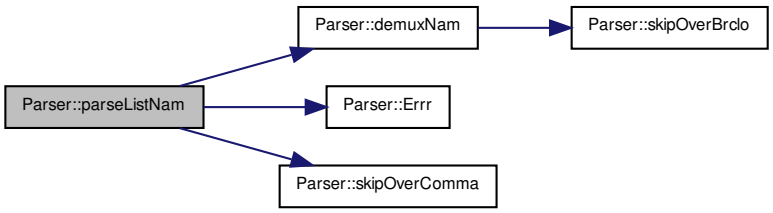
Parameters

*Export_*	the function to call on each find

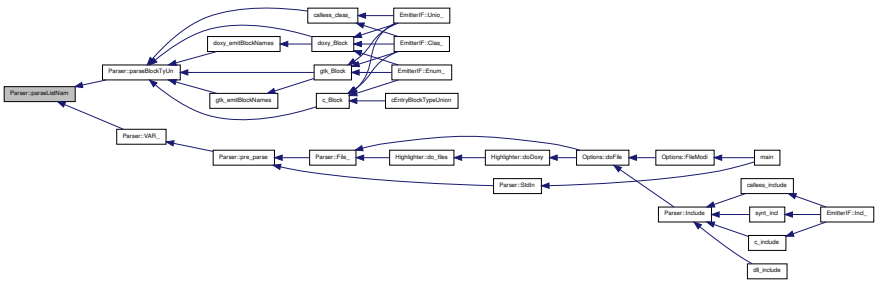
Scan the token list for variable declarations and call the emitter for each find (ie for constructs like DIM AS BYTE Nam1, Nam2(5) = {0,1,2,3,4,5}, Nam3, ...).

Definition at line 303 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.10 SUB Parser::parseListNamTyp (BYVAL_AS_EmitFunc *Export_*)

Evaluate a list of declarations.

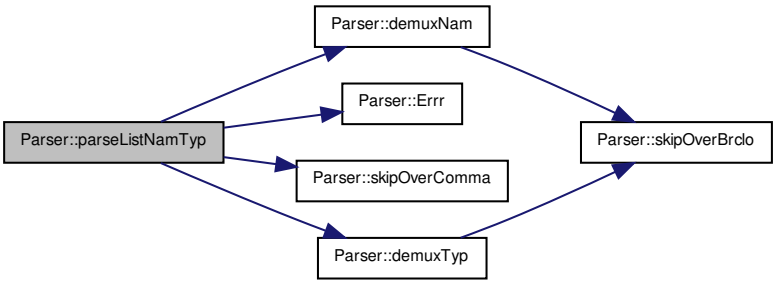
Parameters

*Export_*	the function to call on each find

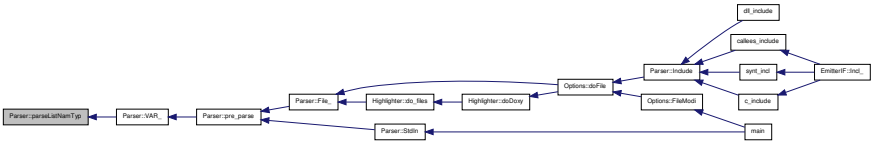
Scan the token list for variable declarations and call the emitter for each find (ie for constructs like DIM Nam1 AS BYTE, Nam2(5) AS UBYTE = {0,1,2,3,4,5}, Nam3 AS STRING, ...).

Definition at line 322 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.11 SUB Parser::parseListPara (BYVAL_AS_EmitFunc *Export_*)

Evaluate a parameter list.

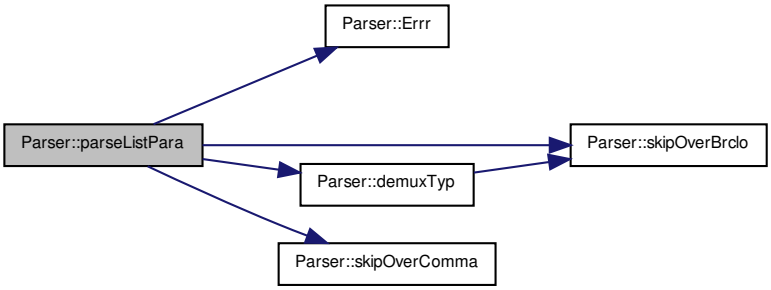
Parameters

*Export_*	the function to call on each find

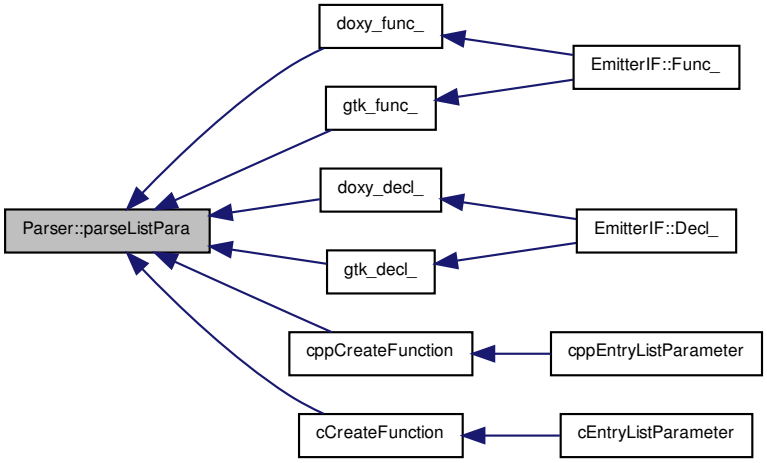
Scan the token list for a parameter list. Evaluate declaration specifiers (BYVAL / BYREF) and handle ellipsis, parameters without name, initializers and empty lists.

Definition at line 436 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.12 SUB Parser::parseBlockEnum (BYVAL_AS_EmitFunc *Export_*)

Evaluate the context of an ENUM block.

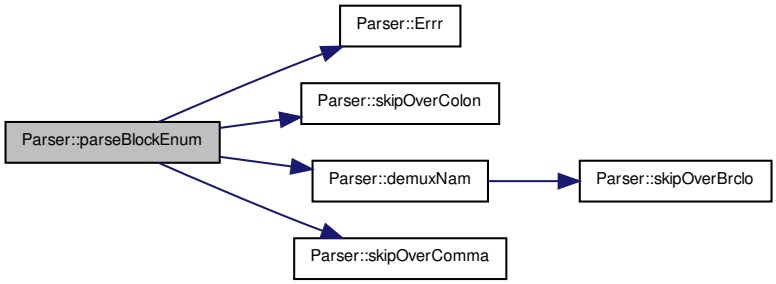
Parameters

*Export_*	the function to call on each find

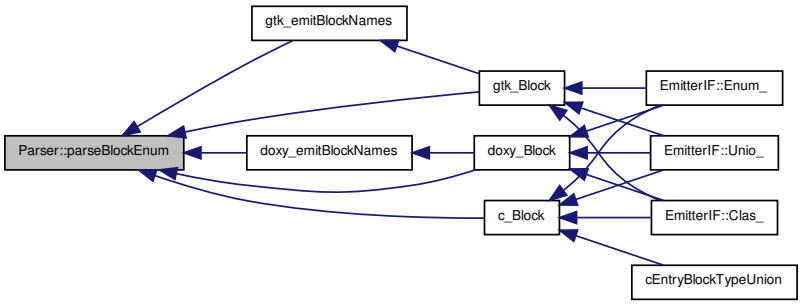
The emitter calls us to evaluate statements inside an ENUM block. So we stop after finding a statement and call the emitter handler one-by-one.

Definition at line 340 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.13 SUB Parser::parseBlockTyUn (BYVAL_AS_EmitFunc *Export_*)

Evaluate the context of a block.

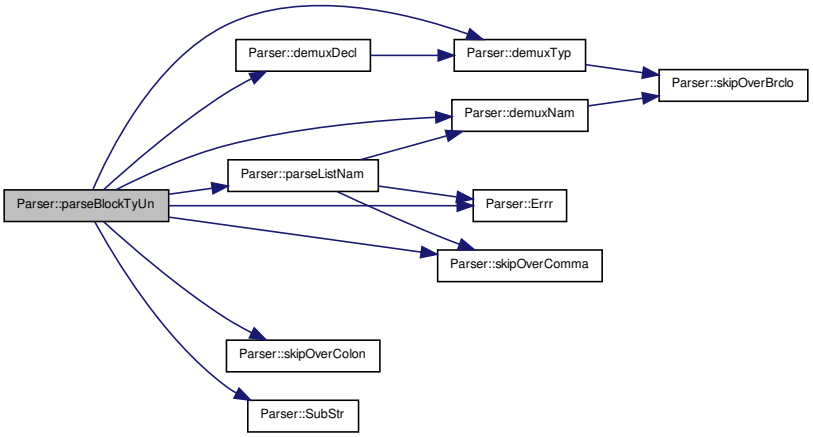
Parameters

*Export_*	the function to call on each find

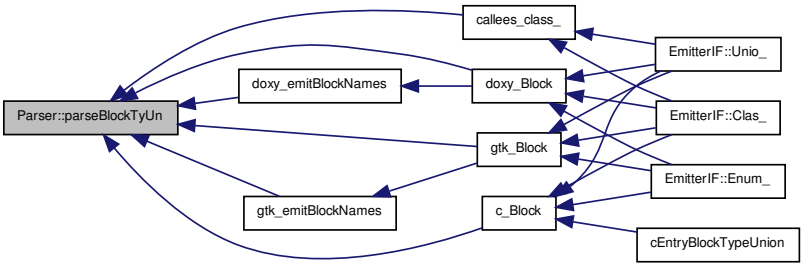
The emitter calls us to evaluate constructs inside a block (TYPE / UNION). So we stop after finding a statement and call the emitter handler one-by-one.

Definition at line 368 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.14 SUB Parser::pre_parse ( ) [private]

Pre-parse all FB source code, search relevant constructs.

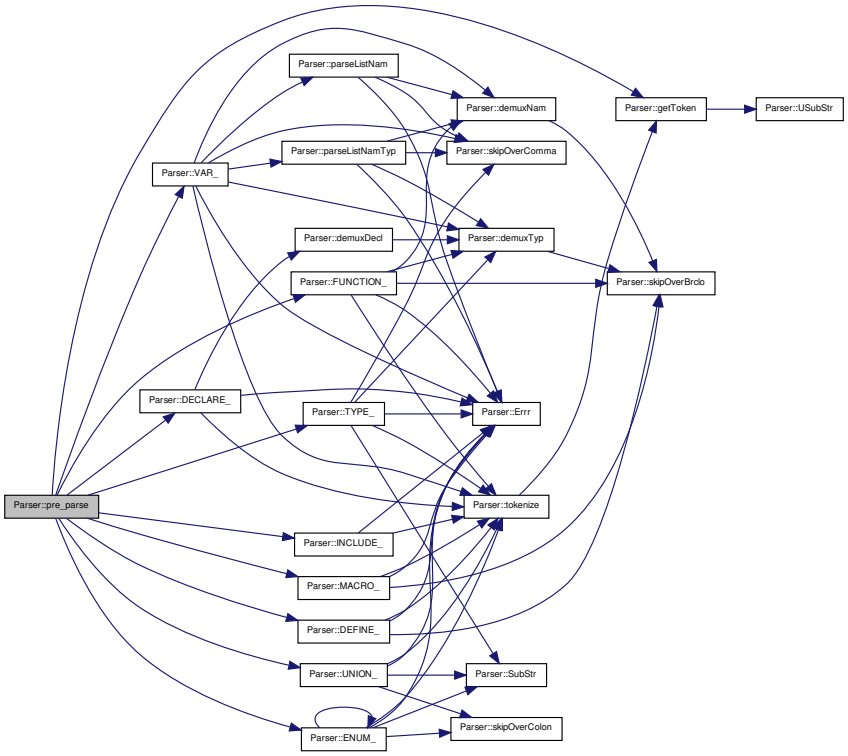
Returns

the length of the output code, if any

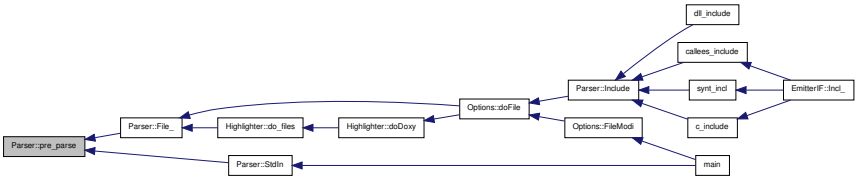
Search the input buffer for a relevant construct. Start detailed parsing on each relevant construct. Otherways skip the current line. Export comments on the way.

Definition at line 1029 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.15 PROPERTY_AS_STRING Parser::USubStr ( ) [private]

Check current token position, extract word if string type.

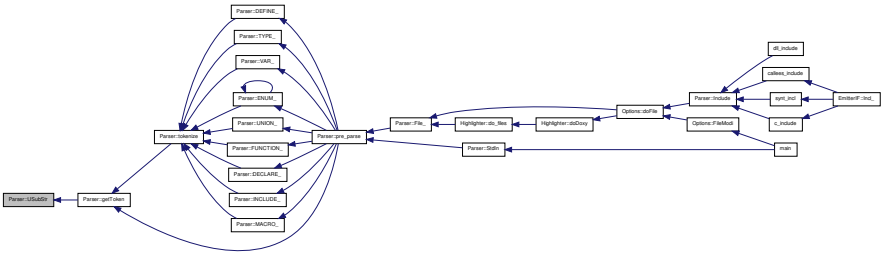
Returns

the current word in upper case

This property returns the context of the current parser position in the input buffer in upper case characters.

Definition at line 1277 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.16 FUNCTION_AS_INTEGER Parser::getToken ( ) [private]

Check word at current parser position.

Returns

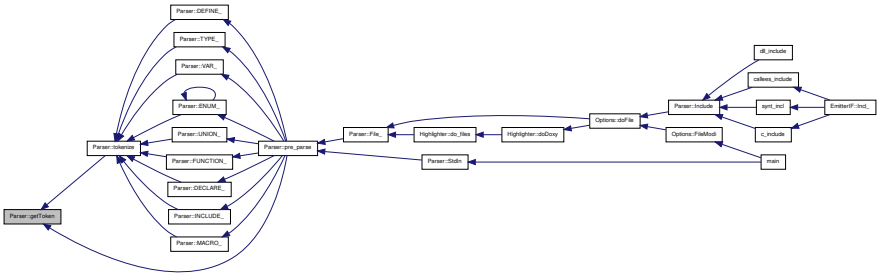
: The token for the word at current position

This function checks the word at the current parser position. In case of a keyword the matching token gets returned. Otherwise Parser::TOK_WORD gets returned.

Definition at line 741 of file fb-doc_parser.bas.

Here is the call graph for this function:

Here is the caller graph for this function:



14.5.4.17 FUNCTION_AS_INTEGER Parser::tokenize (BYVAL_AS_EoS_Modi *Stop_*) [private]

Parse a relevant construct, create a token list.

Parameters

*Stop_*	the condition where to end parsing

Returns

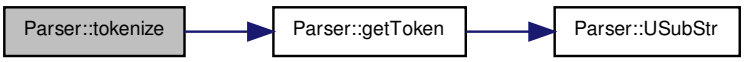
the length of the token list in byte (at least 8)

Start at the current position of the input buffer to check each character. Sort the input in to a token list. The token list contains three values in each entry: the type of the input, its start and its length.

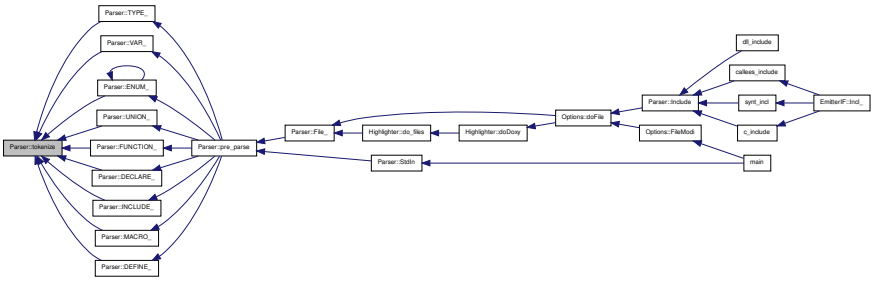
The end of this process gets specified by the Stop_ parameter. It's one of the EoS_Modi enumerators.

Definition at line 947 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.18 FUNCTION_AS_INTEGER Parser::Errr (BYREF_AS_STRING *E*) [private]

Emit an error message.

Parameters

E	the reason for the error message

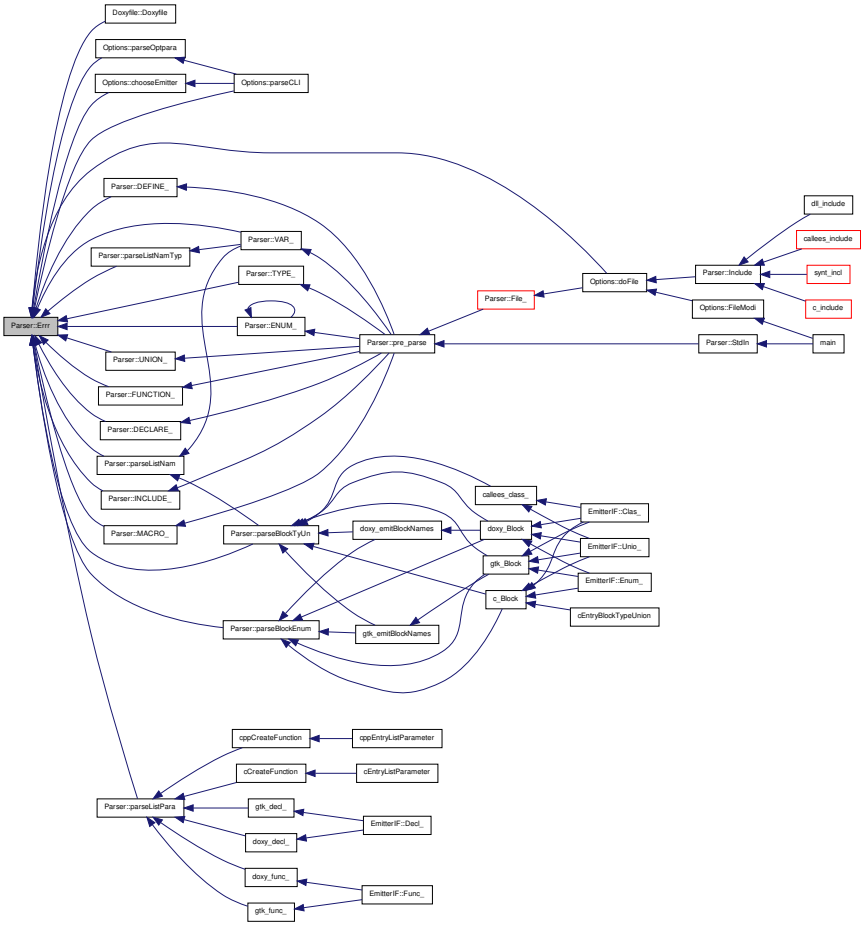
Returns

MSG_ERROR (and MSG_STOP at end of file)

Create an error message and call the error function of the emitter. It depends on the emitter if and where the error is shown.

Definition at line 693 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.19 FUNCTION_AS_INTEGER Parser::demuxNam (BYVAL_AS_INTEGER *MinTk* = TOK_WORD, BYVAL_AS_INTEGER *DeclMod* = 0) [private]

Evaluate a name, dimension and initializer in the token list.

Parameters

MinTk	the token to start at
DeclMod	the modus (normal or declaration)

Returns

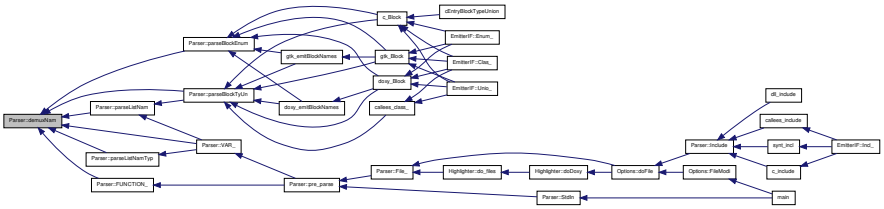
the (doubled) number of steps gone in the token list (or MSG_ERROR on error and MSG_STOP on the end of the token list)

Check the token list for a declaration of a name, starting at the current token (ie behind a VAR keyword). Also checking for parenthesis to specify a dimension and the equal '=' character of an initializer. After execution the current position is at the next token after the equal character, or behind the closing parenthesis, or the name. An error is returned if there is no word at the current position.

Definition at line 112 of file fb-doc_parser.bas.

Here is the call graph for this function:

Here is the caller graph for this function:



14.5.4.20 FUNCTION_AS_INTEGER Parser::demuxTyp (BYVAL_AS_INTEGER *DeclMod* = 0) [private]

Evaluate a type declaration in the token list.

Parameters

DeclMod	the modus (normal or declaration)

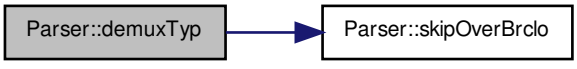
Returns

the (doubled) number of steps gone in the token list (or MSG_ERROR on error and MSG_STOP on the end of the token list)

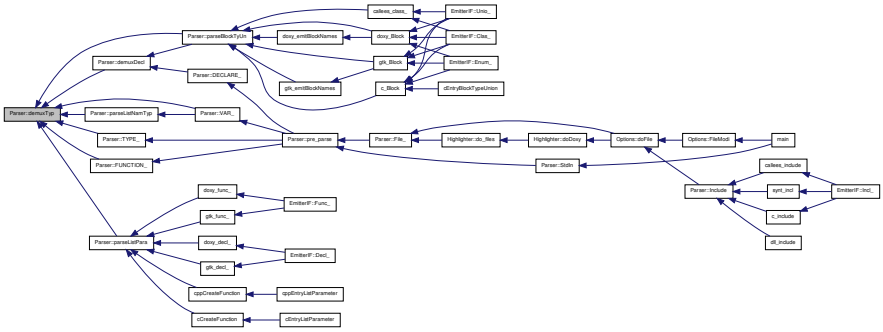
Check the token list for a type declaration. Starting at the AS statement we read the name of the type and all following 'PTR' statements. The type may be a SUB, FUNCTION, PROPERTY, ... After execution the current position is at the next token behind the last PTR, or at the type name, or at the closing parenthesis. An error is returned if there is no word at the current position, or the token list doesn't start with 'AS', or the next token isn't a word.

Definition at line 209 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.21 FUNCTION_AS_INTEGER Parser::demuxDecl ( ) [private]

Evaluate a function declaration in the token list.

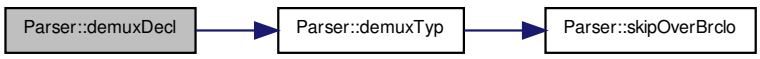
Returns

the the result of demuxTyp() (or MSG_ERROR on error and MSG_STOP on the end of the token list)

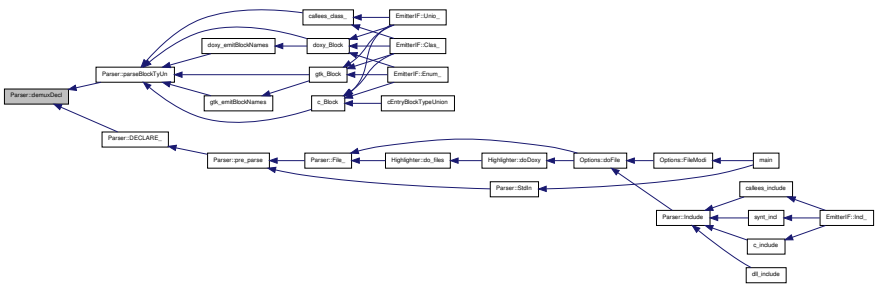
Pre-check the token list for a declaration of a function, starting at the token behind the DECLARE statement. When OK, use demuxTyp() to evaluate the type of the declaration.

Definition at line 167 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.22 FUNCTION_AS_INTEGER Parser::skipOverColon ( ) [private]

Move to the end of statement and one step beyond.

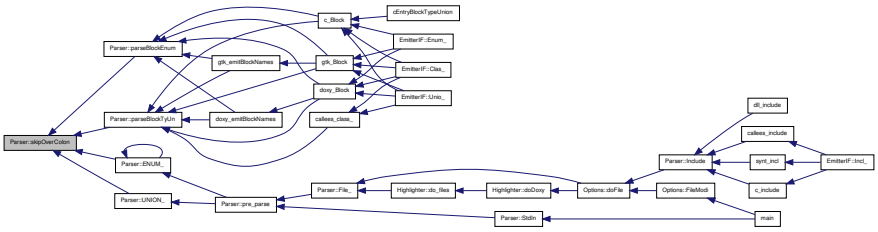
Returns

the (doubled) number of steps gone in the token list (or MSG_ERROR on error and MSG_STOP on the end of the token list)

Step through the token list and search for the end of the current statement. Then walk one step beyond, if not end of token list.

Definition at line 88 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.23 FUNCTION_AS_INTEGER Parser::skipOverBrclo ( ) [private]

Move to the matching right parenthesis and one step beyond.

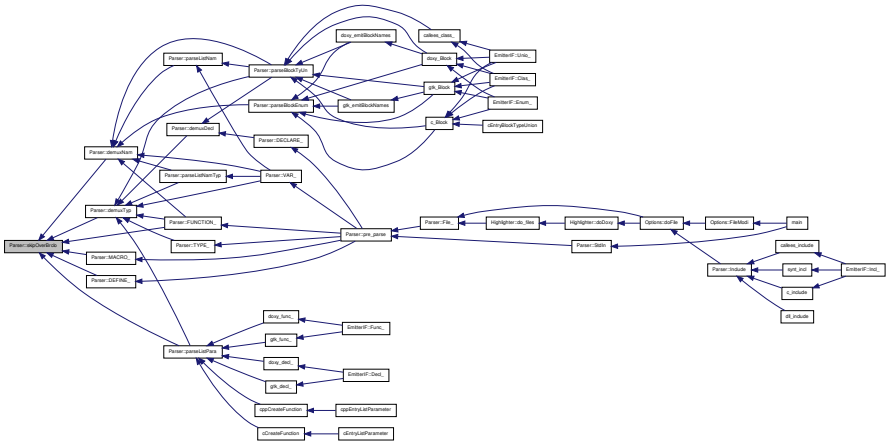
Returns

the (doubled) number of steps gone in the token list (or MSG_ERROR on error and MSG_STOP on the end of the token list)

Step through the token list and search for the next right parenthesis, skipping nested pairs of paranethesis. When we find a matching right parenthesis we walk one step beyond. Else we stop at the next colon or line end.

Definition at line 68 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.24 FUNCTION_AS_INTEGER Parser::skipOverComma ( ) [private]

Move to the next comma and one step beyond.

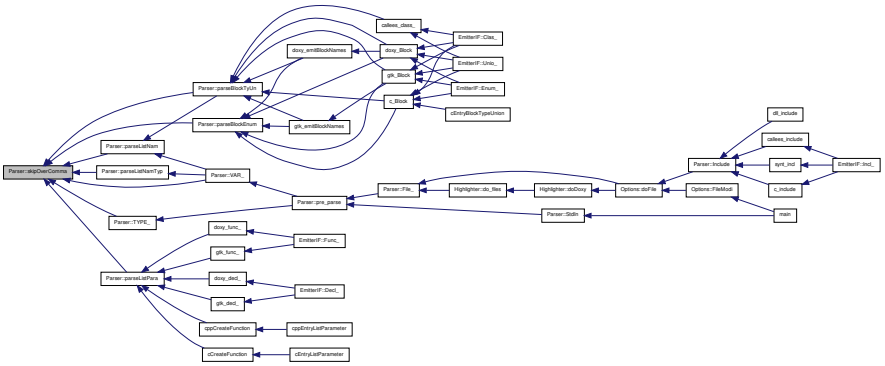
Returns

the (doubled) number of steps gone in the token list (or MSG_ERROR on error and MSG_STOP on the end of the token list)

We step trough the token list and search for the next comma, skipping nested pairs of parenthesis. When we find a comma we walk one step beyond. Otherwise we stop at the next right parenthesis or the next colon or line end.

Definition at line 45 of file fb-doc_parser.bas.

Here is the caller graph for this function:



14.5.4.25 FUNCTION_AS_INTEGER Parser::VAR_ ( ) [private]

Evaluate a variable declaration.

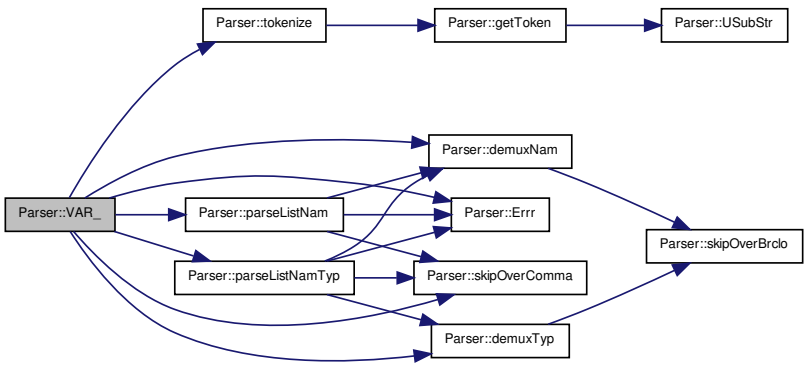
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

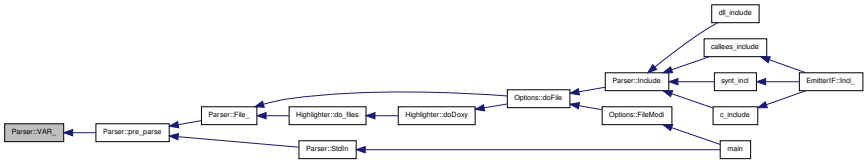
The pre-parser found a VAR, DIM, CONST, EXTERN, COMMON or STATIC keyword. We tokenize the current line and send the result to the emitter, or we call Errr() on syntax problems.

Definition at line 512 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.26 FUNCTION_AS_INTEGER Parser::TYPE_ ( ) [private]

Evaluate a TYPE or CLASS statement.

Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

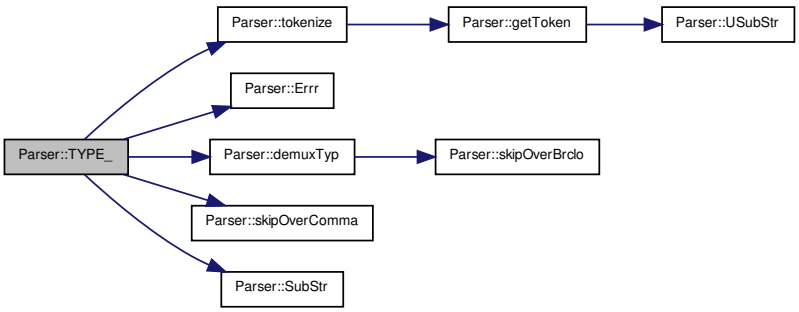
The pre-parser found a TYPE or CLASS keyword. In case of TYPE we tokenize the line and check if it is an alias declaration. In that case we call the emitter for a declaration on that single line. Otherwise we tokenize the complete block and call the emitter for a TYPE block. Or we call Errr() on syntax problems.

Note: the C emitter creates

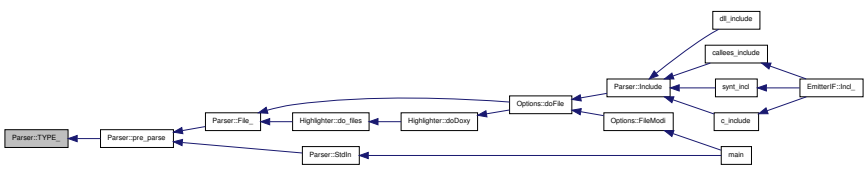
- 'typedef Typ Nam' for 'TYPE AS Typ Nam'
- 'typedef struct Typ Nam' for 'TYPE Nam AS Typ'

Definition at line 476 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.27 FUNCTION_AS_INTEGER Parser::ENUM_ ( ) [private]

Evaluate an ENUM block.

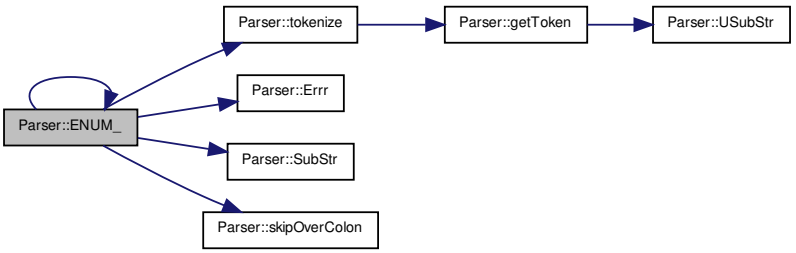
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

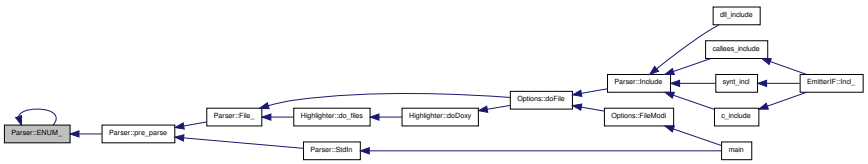
The pre-parser found an ENUM keyword. We tokenize the context of the complete block and call the emitter, or we call Errr() on syntax problems.

Definition at line 548 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.28 FUNCTION_AS_INTEGER Parser::UNION_ ( ) [private]

Evaluate an UNION block.

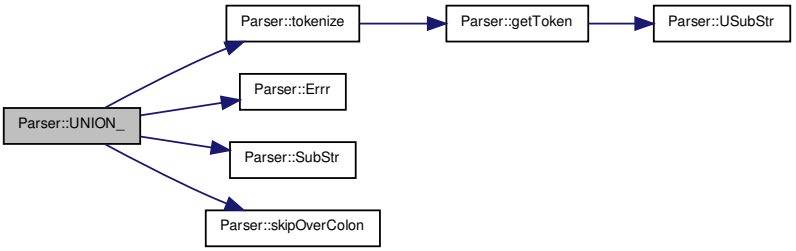
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

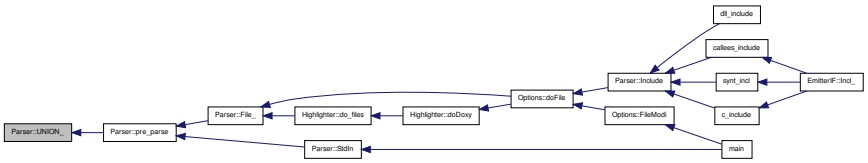
The pre-parser found an UNION keyword. We tokenize the context of the complete block and call the emitter, or we call Errr() on syntax problems.

Definition at line 566 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.29 FUNCTION_AS_INTEGER Parser::FUNCTION_ ( ) [private]

Evaluate a function.

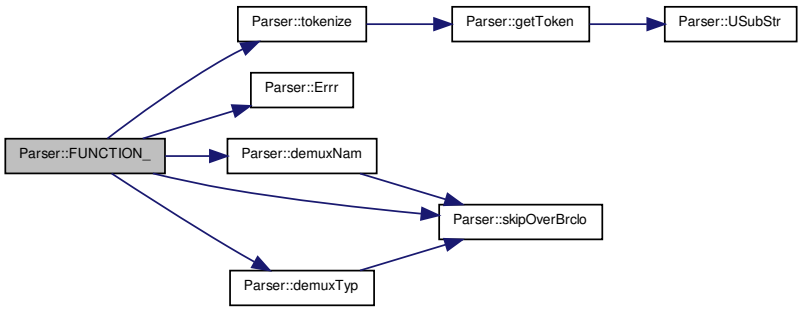
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

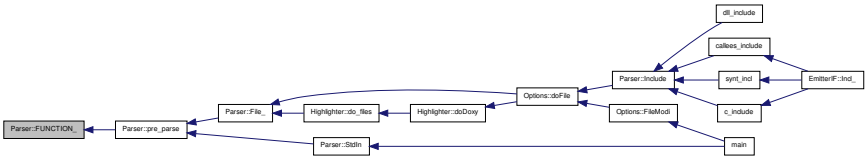
The pre-parser found a SUB, FUNCTION, CONSTRUCTOR, DESTRUCTOR, PROPERTY or OPERATOR keyword. We tokenize the complete block, evaluate the name and type and send the result to the emitter, or we call Errr() on syntax problems.

Definition at line 586 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.30 FUNCTION_AS_INTEGER Parser::DECLARE_ ( ) [private]

Evaluate a forward declaration.

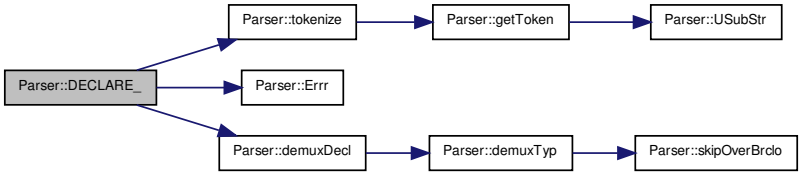
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

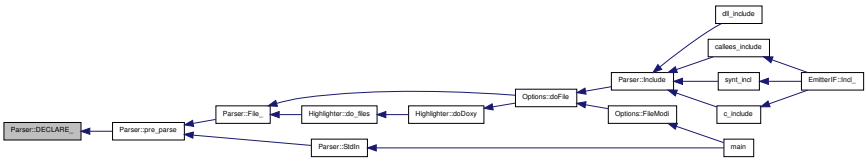
The pre-parser found a DECLARE keyword. We tokenize the current line and send the result to the emitter, or we call Errr() on syntax problems.

Definition at line 619 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.31 FUNCTION_AS_INTEGER Parser::INCLUDE_ ( ) [private]

Evaluate an #INCLUDE line.

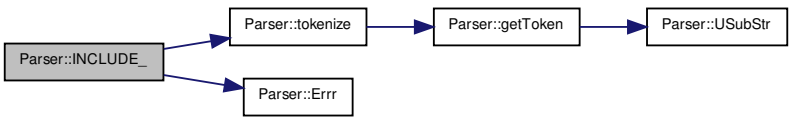
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

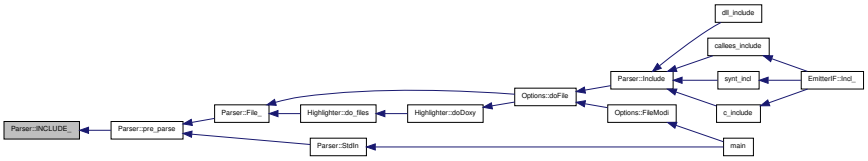
The pre-parser found an #INCLUDE line. We tokenize the line and call the emitter, or we call Errr() handler on syntax problems. It's up to the emitter function to follow the source tree (= create a new parser and load the file).

Definition at line 636 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.32 FUNCTION_AS_INTEGER Parser::DEFINE_ ( ) [private]

Evaluate a #DEFINE declaration.

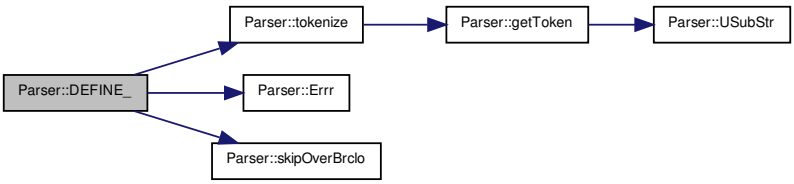
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

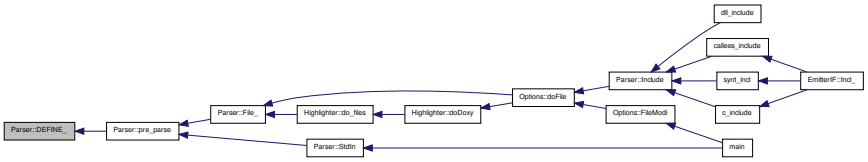
The pre-parser found a #DEFINE line. We tokenize the line and call the emitter, or we call Errr() handler on syntax problems.

Definition at line 671 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.4.33 FUNCTION_AS_INTEGER Parser::MACRO_ ( ) [private]

Evaluate a #MACRO declaration.

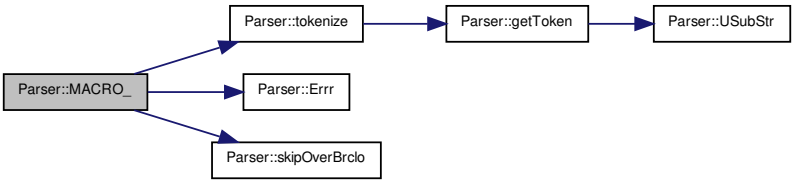
Returns

MSG_ERROR (or MSG_STOP on the end of the token list)

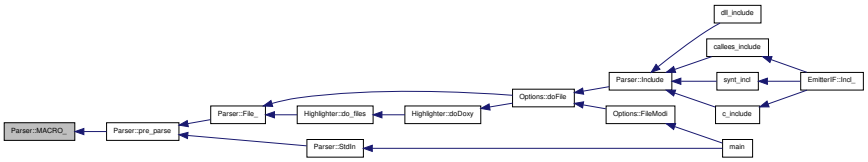
The pre-parser found a #MACRO block. We tokenize the block and call the emitter, or we call Errr() handler on syntax problems.

Definition at line 652 of file fb-doc_parser.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



14.5.5 Member Data Documentation

14.5.5.1 STRING Parser::InPath = ""

path of current of input file (option --tree)

Definition at line 148 of file fb-doc_parser.bi.

14.5.5.2 STRING Parser::Buf

the input buffer

Definition at line 148 of file fb-doc_parser.bi.

14.5.5.3 STRING Parser::ErrMsg

an error message

Definition at line 148 of file fb-doc_parser.bi.

14.5.5.4 STRING Parser::Tok

the token list (binaries)

Definition at line 148 of file fb-doc_parser.bi.

14.5.5.5 STRING Parser::BlockNam

the name of a block (ENUM, UNION, TYPE)

Definition at line 148 of file fb-doc_parser.bi.

14.5.5.6 INTEGER_PTR Parser::StaTok

the pre-parsed token

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.7 INTEGER_PTR Parser::NamTok

the name token of the construct

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.8 INTEGER_PTR Parser::DimTok

the token of the left parenthesis of a dimension

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.9 INTEGER_PTR Parser::IniTok

the start token of an initializer ('=')

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.10 INTEGER_PTR Parser::BitTok

the start token of an bitfiled declaration (':')

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.11 INTEGER_PTR Parser::TypTok

the token of the type keyword

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.12 INTEGER_PTR Parser::ShaTok

the token of the SHARED keyword

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.13 INTEGER_PTR Parser::PtrTok

the token of the first PTR keyword

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.14 INTEGER_PTR Parser::Co1Tok

the token of the first CONST keyword (if any)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.15 INTEGER_PTR Parser::Co2Tok

the token of the second CONST keyword (if any)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.16 INTEGER_PTR Parser::FunTok

type of the pre-parsed token

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.17 INTEGER_PTR Parser::CalTok

the token of the calling convention keyword (if any)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.18 INTEGER_PTR Parser::AliTok

the token of the ALIAS keyword (if any)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.19 INTEGER_PTR Parser::As_Tok

the token of the AS keyword (if no SUB)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.20 INTEGER_PTR Parser::ParTok

the token of the left parenthesis of a parameter list

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.21 INTEGER_PTR Parser::By_Tok

the token of the declaration specifier (BYVAL / BYREF)

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.22 INTEGER_PTR Parser::DivTok

the token of different purposes like the STATIC keyword in a member function declaration (if any) or the LIB keyword in a normal declaration

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.23 INTEGER_PTR Parser::Tk1

the first token in a statment

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.24 INTEGER_PTR Parser::EndTok

the last token in the list

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.25 INTEGER_PTR Parser::UserTok

a token for customised usage in emitter-handlers

Definition at line 175 of file fb-doc_parser.bi.

14.5.5.26 INTEGER Parser::Po

the current position in the input buffer Parser::Buf

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.27 INTEGER Parser::Fin

the last position in the input buffer Parser::Buf

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.28 INTEGER Parser::PtrCount

the number of PTR keywords

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.29 INTEGER Parser::SrcBgn

start position to export FB source

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.30 INTEGER Parser::LineNo

the current line in the input buffer Parser::Buf

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.31 INTEGER Parser::LevelCount

the level in nested blocks (one based)

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.32 INTEGER Parser::InTree

flag to indicate if to follow source tree #INCLUDEs

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.33 INTEGER Parser::ListCount

the current entry in a list (zero based)

Definition at line 206 of file fb-doc_parser.bi.

14.5.5.34 INTEGER Parser::ToLast [private]

the type of the token before the current one

Definition at line 288 of file fb-doc_parser.bi.

14.5.5.35 INTEGER_PTR Parser::Tk [private]

the current token of the parser

Definition at line 291 of file fb-doc_parser.bi.

14.5.5.36 INTEGER_PTR Parser::A [private]

start of a word in pre-parsing

Definition at line 291 of file fb-doc_parser.bi.

14.5.5.37 INTEGER_PTR Parser::L [private]

length of a word in pre-parsing

Definition at line 291 of file fb-doc_parser.bi.

14.5.5.38 EmitterIF_PTR Parser::Emit [private]

the emitter interface

Definition at line 295 of file fb-doc_parser.bi.

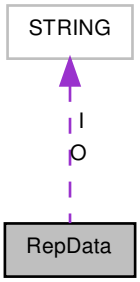
The documentation for this class was generated from the following files:

- src/fb-doc_parser.bi
- src/fb-doc_parser.bas

14.6 RepData Class Reference

A container for string replacements.

Collaboration diagram for RepData:



Public Member Functions

- FUNCTION_AS_ZSTRING_PTR add (BYREF_AS_STRING, BYREF_AS_STRING)

  Add a new element pair.
- FUNCTION_AS_ZSTRING_PTR rep (BYREF_AS_STRING)

  Search for a string, get its replacement (if any)

Public Attributes

- STRING O

  The output (a list of counters and strings)
- STRING I = MKI(0) & CHR(1)

  The input (a list of strings to search for)

14.6.1 Detailed Description

A container for string replacements.

This class is to store two tables, one of search strings and a second one of their replacements. It's used to collect the links from the original source and replace symbol names in the emitter output, as well as referenced line numbers and #INCLUDE files.

Definition at line 27 of file fb-doc_emit_syntax.bas.

14.6.2 Member Function Documentation

14.6.2.1 FUNCTION_AS_ZSTRING_PTR RepData::add (BYREF_AS_STRING *S*, BYREF_AS_STRING *R*)

Add a new element pair.

Parameters

S	The string to search for
R	The string to replace with

Returns

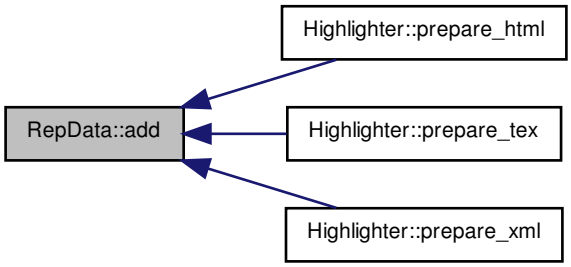
zero on success, otherwise a ZSTRING PTR to an error code

Add a new pair of strings to the container tables. The function checks if the search string contains illegal characters or if it's already defined. In this cases nothing happens and an error message gets returned. Otherwise the search string and its replacement gets stored in the container.

The search string must not contain characters in the range CHR(0) to CHR(2).

Definition at line 50 of file fb-doc_emit_syntax.bas.

Here is the caller graph for this function:



14.6.2.2 FUNCTION_AS_ZSTRING_PTR RepData::rep (BYREF_AS_STRING *S*)

Search for a string, get its replacement (if any)

Parameters

S	The string to search for

Returns

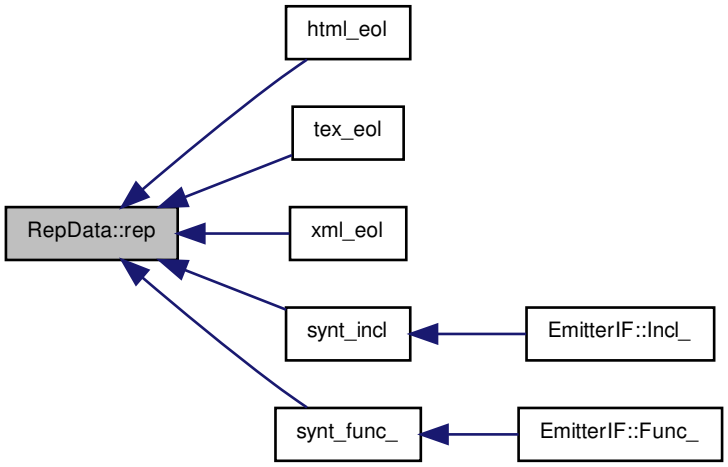
A pointer to the replacement (if any)

This function searches for the string given as parameter. If the string is in the search words, a pointer to it's replacement gets returned. Otherwise a pointer to the original string gets returned.

The function searches case sensitive.

Definition at line 69 of file fb-doc_emit_syntax.bas.

Here is the caller graph for this function:



14.6.3 Member Data Documentation

14.6.3.1 STRING RepData::O

The output (a list of counters and strings)

Definition at line 29 of file fb-doc_emit_syntax.bas.

14.6.3.2 STRING RepData::I = MKI(0) & CHR(1)

The input (a list of strings to search for)

Definition at line 30 of file fb-doc_emit_syntax.bas.

The documentation for this class was generated from the following file:

- src/fb-doc_emit_syntax.bas

Chapter 15

File Documentation

15.1 src/Development.md File Reference

15.2 src/Development.md

```
00001 Further Development  {#pageToDo}
00002 ==================
00003
00004 In combination with Doxygen fb-doc is a powerful documentation
00005 system for FreeBasic source code. The Doxygen support for FreeBasic
00006 is much better than for any other BASIC dialect (ie like VB6). But
00007 there's still some room for optimization, for Doxygen support as
00008 well as for other features, like:
00009
00010  - additional emitters to support further documentation systems or other tools
00011  - a hash table for callees to speed up the execution
00012  - currently fb-doc only works with clean FB code. When one of the
00013    used standard keywords get \#`UNDEF`ined, grazy things may happen.
00014  - ...
00015
00016 Feel free to post your ideas, bug reports, wishes or patches, either
00017 to the project page at
00018
00019  - \ProjPage
00020
00021 or by email to
00022
00023  - see section \ref licence
```

15.3 src/Emitters.md File Reference

15.4 src/Emitters.md

```
00001 Emitters in Detail  {#pageEmitterDetails}
00002 ==================
00003 \tableofcontents
00004
00005 \section sectEmitterCSource C Source
00006
00007 | Emitter                | C_Source                                          |
00008 | ----------------------: | :------------------------------------------------ |
00009 | `-a` *or* `--asterix`   | prepend "* " to each line in multi line comment |
00010 | `-c` *or* `--cstyle`    | emit real C types, \#`include "*.c" "*.h"`      |
00011 | `-t` *or* `--tree`      | follow source tree \#`INCLUDES`                 |
00012
00013 This emitter translates its input in to the intermediate format,
00014 usually used by the C back-end. It's not a real compiler, just the
00015 documentation relevant constructs get emitted. This is declarations and
00016 a few further constructs.
00017
00018 |            Construct | Keywords                                                |
00019 | -------------------: | :------------------------------------------------------ |
00020 |            variables | `VAR  DIM  CONST  COMMON  EXTERN  STATIC`               |
```

```
00021 |              blocks | 'ENUN  UNION  TYPE  CLASS'                                |
00022 | forward declarations | 'TYPE  TYPE alias'                                        |
00023 |            functions | 'SUB  FUNCTION  PORPERTY  CONSTRUCTOR  DESTRUCTOR'       |
00024 |              macros | \#'DEFINE'  \#'MACRO'                                     |
00025
00026 It doesn't handle SCOPE nor NAMESPACE blocks yet.
00027
00028 The output contains translated C source code and the special comments
00029 for the FB source. Everything is at the same line number and in the
00030 original order.
00031
00032 The translation can either contain type declarations in the mangeled
00033 (FB-like) style or real types in C syntax. The first is default and the
00034 second is set by option '--cstyle'. This option also influences the
00035 translation of 'TYPE's ('class{public:' *or* 'typedef struct{'),
00036 'CONSTRUCTOR's and 'DESTRUCTOR's ('name::name()' *or* 'void()') and the
00037 names in \#'include' statements (original *or* suffix '".c" ".h"').
00038
00039 Option '--asterix' makes fb-doc to start each line in a special multi
00040 line comment block by the characters '* '. This is mandatory for the
00041 gtk-doc back-end. Adding these characters to the FB source code has two
00042 downsides: it blows up the code size and these characters complicate
00043 automatic formating of the context in an editor.
00044
00045 The emitter is designed to be used in standard mode (Doxygen filter)
00046 and in mode '--file-mode' to generate the intermediate format for the
00047 back-end.
00048
00049 Furthermore the output can be helpful when you intend to write a
00050 library in FB and later use this library in a C project. This emitter
00051 can translate the FB headers to real C headers by using option
00052 '--cstyle'. You just have to translate the \#'define' lines manually.
00053
00054
00055 \section sectEmitterGtkTempl GtkDoc Templates
00056
00057 | Emitter                          | GtkDocTemplates                    |
00058 | --------------------------------: | :--------------------------------- |
00059 | -a *or* --asterix                |                                    |
00060 |                                  |                                    |
00061 |                                  |                                    |
00062
00063 This emitter is designed to generate templates for documenting FB code
00064 with gtk-doc tool chain (back-end). It generates output containing the
00065 all original code. Documentation relevant parts of the code get
00066 prepended by a template for gtk-doc. This template contains the symbol
00067 names extracted from the source code and 'FIXME' marks for the
00068 individual text fragments.
00069
00070 The emitter is designed to be used in mode '--geany-mode'. Installed as
00071 a custom command in Geany IDE it receives selected source code and
00072 returns it with one or more the templates added.
00073
00074 It doesn't really make sense to use this emitter in modes '--file-mode',
00075 '--list-mode' or '--syntax-mode', since their output files are used for
00076 other context.
00077
00078 Further usage may be generating templates for a complete file
00079
00080 \section sectEmitterDoxyTempl Doxygen Templages
00081
00082 | Emitter                          | DoxygenTemplates                   |
00083 | --------------------------------: | :--------------------------------- |
00084 | -a *or* --asterix                | no effect                          |
00085 | -                                |                                    |
00086 |                                  |                                    |
00087
00088 This emitter is designed to
00089
00090 The emitter is designed to be used in mode '--'
00091
00092 It doesn't really make sense to use this emitter in modes '--file-mode',
00093 '--list-mode' or '--syntax-mode', since their output files are used for
00094 other context.
00095
00096 Further usage may be
00097
00098
00099 \section sectEmitterLfn List of Function Names
00100
00101 | Emitter                | FunctionNames                  |
00102 | ----------------------: | :------------------------------ |
00103 | '-a' *or* '--asterix'  | no effect                      |
00104 | '-c' *or* '--cstyle'   | no effect                      |
00105 | '-t' *or* '--tree'     | follow source tree \#'INCLUDES' |
00106
00107 This emitter outputs the function names from FB source code. It emitts
```

```
00108 a list with one name per line for `SUB`s, `FUNCTION`s and `PROPERTY`s.
00109 In case of an UDT member the UDT name also gets emitted.
00110
00111 It is designed to generate the list of function names file *fb-doc.lfn*
00112 in mode `--list_mode`. This file is used for caller / callee graphs and
00113 gets read by the emitter `C_Source`. In this mode the output in the file
00114 starts with a header line and the list begins in the second line. The
00115 list contains line ends in UNIX style (no caridge return, just a line
00116 feed). This must not be changed!
00117
00118 In other run modes no header line gets emitted. The list starts with
00119 the first function name found in the input. There is no separator
00120 between the output from different input files. But when used in mode
00121 `--file-mode` the output is one file per input file.
00122
00123
00124 \section sectEmitterSyntax Syntax Hightlighting
00125
00126 | Emitter                 | SyntaxHighLighting              |
00127 | ----------------------: | :------------------------------ |
00128 | `-a` *or* `--asterix`   | no effect                       |
00129 | `-c` *or* `--cstyle`    | no effect                       |
00130 | `-t` *or* `--tree`      | follow source tree \#`INCLUDES` |
00131
00132 This emitter outputs the the original source code, wherein the FB
00133 syntax is highlighted by enclosing tags. In mode `--syntax-mode`
00134 three different tag formats can get generated
00135
00136 |      file type | tags in `--syntax-mode`                    |
00137 | -------------: | :----------------------------------------- |
00138 | Html (default) | `<span class="...">CODE</span>`            |
00139 |          LaTeX | `\textcolor{...}{CODE}`                    |
00140 |            XML | `<highlight class="...">CODE</highlight>` |
00141
00142 In other run modes the emitter generates Html output.
00143
00144 Each line starts with a line number. All input gets separated in to
00145 these categories (replacing the '...' in the above tag format examples)
00146
00147 |     style class | used for                                                                  |
00148 | --------------: | :------------------------------------------------------------------------ |
00149 |          `line` | normal code no special highlighting                                       |
00150 |       `keyword` | FB keywords like `FOR, NEXT, DECLARE, ...`                                |
00151 |   `keywordtype` | inbuild types like `UBYTE, INTEGER, ...` and also `BYREF, AS, ...`        |
00152 |  `preprocessor` | preprocessor statements like \#`if`, \#`INCLUDE`, ...                     |
00153 | `stringliteral` | string literals like `"Hello world"`                                      |
00154 |       `comment` | comments (multi line /&amp;apos; ... &amp;apos;/ or single line &amp;apos; ...) |
00155 |        `lineno` | style class for the line number (starting at 1)                           |
00156
00157 Doxygen supports a further style class named `keywordflow`, meant to be
00158 used for keywords like `WHILE ... WEND`, `DO ...LOOP` and so on. This
00159 doesn't work for FB code since some keywords are used in different
00160 meanings (ie `FOR ... next` and also `open(... FOR input)`). So fb-doc
00161 doesn't use this style class.
00162
00163 This emitter is designed to be used in mode `--syntax_mode` to repair
00164 the syntax hightlighting in Doxygen output. In this mode fb-doc reads
00165 the original Doxygen output files and extracts the links to the
00166 documentation. The header and footer of the original file get copied to
00167 the new file and the listing section get filled with fresh code,
00168 including the highlighting tags and also the links from the
00169 original Doxygen output.
00170
00171 In other modes like `--geany-mode` or `--list-mode` the emitter works
00172 in its default mode. Its output contains plain Html code using the
00173 above mentioned style tags.
00174
00175 Line numbers start at one, in mode `--file-mode` for each file. In
00176 other modes when the output is generated from several input files, the
00177 line numbers increase continuously.
00178
00179 To use the output in any Html page it needs a header (or file)
00180 defining the above mentioned style classes. As an example check any
00181 Doxygen output (ie the Html tree of this documentation and the file
00182 *fb-doc.css*).
```

15.5 src/Examples.md File Reference

15.6 src/Examples.md

```
00001 Examples  {#pageExamples}
```

```
00002 ========
00003 \tableofcontents
00004
00005 Here're some examples for using fb-doc and its features.
00006
00007
00008
00009 \section sectExaCli Command Line Interface
00010
00011 In a first glance you can execute fb-doc on its own source code in
00012 folder *src* to learn about its usage. Extract the archive *fb-doc.zip*
00013 to any folder and compile (`fbc -w all fb-doc.bas`). Then you can
00014 immediately test the binary in that folder.
00015
00016 \note The following examples are for UNIX like systems. Omit the
00017       leading `./` when you're on other systems.
00018
00019 -# To output the version info and help text execute in a terminal
00020    ~~~{.sh}
00021    ./fb-doc --version
00022    ./fb-doc --help
00023    ~~~
00024 -# To see the intermediate format for a certain file (fb-doc.bas in this
00025    case) execute
00026    ~~~{.sh}
00027    ./fb-doc fb-doc.bas
00028    ~~~
00029 -# Instead of watching the output in the terminal you can write it to a
00030    file using
00031    ~~~{.sh}
00032    ./fb-doc fb-doc.bas > fb-doc.c
00033    ~~~
00034    and compare both files. The file *fb-doc.c* contains just a part of
00035    the original code (variable declarations, some special comments,
00036    \#`INCLUDE`s and an empty function `main()`). Each context is at the
00037    same line number.
00038 -# To test the output of the emitter `SyntaxHighLighting` execute
00039    ~~~{.sh}
00040    ./fb-doc --emitter "SyntaxHighLighting" fb-doc.bas > fb-doc.html
00041    ~~~
00042    and the new file *fb-doc.html* contains Html code with highlighting
00043    tags for that FB source file. (You won't see highlighted code when
00044    you load this document in to a browser, because the class
00045    declaratinons for the highlighting tags are missing.)
00046 -# Instead of option `--emitter` you can use the short form `-e`. And
00047    instead of a single file name you can use a file pattern. Here we
00048    test the emitter `FunctionNames` with terminal output
00049    ~~~{.sh}
00050    ./fb-doc -e "FunctionNames" "*.bi"
00051    ~~~
00052    and get the name `null_emitter` as single output line, which is the
00053    only function declared in the code files matching this pattern.
00054 -# When we use `"fb-doc.bas"` as file specification we'll get empty
00055    output since this file contains no function declaration. But when we
00056    add option `--tree` fb-doc follows each \#`INCLUDE` statement in the
00057    source tree and generates a list of all function names.
00058    ~~~{.sh}
00059    ./fb-doc -e "FunctionNames" --tree "fb-doc.bas"
00060    ~~~
00061    The list in the terminal output also contains some messages about
00062    the files scanned during the tree run.
00063 -# When you pipe the output to a file, the messages will be
00064    shown in the terminal and the file contains the names list only.
00065    ~~~{.sh}
00066    ./fb-doc -e "FunctionNames" -t "fb-doc.bas" > fb-doc.lfn
00067    ~~~
00068    This list includes the function names of the fb-doc source tree, but
00069    it doesn't contain the function names from the file `Plugin.bas`.
00070 -# To generate a list of all function names you can specify several
00071    file patterns and collect the function names of all source files in
00072    in a single file, like
00073    ~~~{.sh}
00074    ./fb-doc -e "FunctionNames"  "*.bas" "*.bi" > fb-doc.lfn
00075    ~~~
00076    Option `--recursiv` will make fb-doc searching in subfolders also.
00077
00078 -# Since this list is often needed to generate caller / callee graphs
00079    in Doxygen output, fb-doc has the special run mode \ref
00080    subsectOptListMode to create it. Use option `--list-mode` (short
00081    form `-l`) to activate it and execute fb-doc near the Doxyfile to
00082    get the file at the right place
00083    ~~~{.sh}
00084    cd ../doc
00085    ./src/fb-doc -l
00086    ~~~
00087    fb-doc scans the *Doxyfile* for
00088    the `INPUT` path and the `RECURSIV` setting, uses patterns `"*.bas"
```

```
00089    "*.bi"` to scan for input files and writes the output to the
00090    file *fb-doc.lfn* in the current folder (overriding an existing, if
00091    any, without warning)
00092 -# When you installed *Doxygen* and *GraphViz* you're ready to generate
00093    your first documentation from FB source. Just execute in the folder
00094    *doc*
00095    ~~~{.sh}
00096    doxygen
00097    ./src/fb-doc -s
00098    ~~~
00099    The first command generates a lot of Doxygen messages (depending on
00100   the settings in *Doxyfile*) and will run some seconds. It should end
00101   with message ''*** Doxygen has finished''. The second command makes
00102   fb-doc run in \ref subsectOptSyntaxMode and repair the listing
00103   sections in the Doxygen output files. After all you should have a
00104   new folder *html* (in the folder *doc*) containing your personal
00105   version of this documentation. Test it by loading
00106   `..doc/html/index.html` in to your browser.
00107
00108 Under the bottom line 1. to 8. are informal examples. You need steps 9.
00109 and 10. for your personal project. It should have
00110
00111 - well prepared FB source files (supported by `--geany-mode "DoxygenTemplates"`) and
00112 - a matching Doxyfile (using fb-doc as input filter)
00113
00114 to auto-generate the documentation by executing
00115
00116 ~~~{.sh}
00117 cd ../doc
00118 fbdoc -l
00119 doxygen
00120 fbdoc -s
00121 ~~~
00122
00123 \note This conclusion assumes a complete installation of *fb-doc*, *Doxygen* and *GraphViz*.
00124
00125
00126 \section sectExaComments Comments (in general)
00127
00128 The C back-ends work on an intermediate format. This format contains
00129 comments exported from the FB source. fb-doc doesn't export all
00130 comments, since the programmer may want to have private comments as
00131 well (ie ToDo marks). On the other hand fb-doc provides a way to export
00132 context directly to the intermediate format, to do tricky things with
00133 the back-end in use.
00134
00135 To export a comment to the intermediate format it needs to be marked as
00136 a special comment by starting its context with a magic character
00137
00138 - `*` for comments (exported as C comment)
00139 - `&` for direct exports (exported as C source)
00140
00141 \note In these examples the ends of the FreeBasic multi line comments
00142       are fakes, because the straight single quote mark cannot be used
00143       in the source (Geany lexer gets confused). Instead an acute
00144       accent is used. Correct them when you intend to copy / paste the
00145       code.
00146
00147 -# A line end comment in the FB source like
00148    ~~~{.bas}
00149    '* a line end comment
00150    ~~~
00151    gets emitted as
00152    ~~~{.c}
00153    / /! a line end comment
00154    ~~~
00155 -# And a multi line comment in FB source like
00156    ~~~{.bas}
00157    /'*
00158    Some
00159    lines of
00160    comment '/
00161    ~~~
00162    gets emitted as
00163    ~~~{.c}
00164    /**
00165    Some
00166    lines of
00167    comment * /
00168    ~~~
00169    or &mdash; when option `--asterix` is set &mdash; as
00170    ~~~{.c}
00171    /**
00172    * Some
00173    * lines of
00174    * comment * /
00175    ~~~
```

```
00176 -# Direct exports are only available for line end comments like
00177    ~~~{.bas}
00178    '&/ / a line end comment in C syntax
00179
00180    '&/*
00181    '&Some
00182    '&lines of
00183    '&C comment
00184    '&* /
00185    ~~~
00186    get emitted as
00187    ~~~{.c}
00188    / / a line end comment in C syntax
00189
00190    /*
00191    Some
00192    lines of
00193    C comment
00194    * /
00195    ~~~
00196 -# Normal FreeBasic comments (without magic character) wont be emitted.
00197    ~~~{.bas}
00198    ' a line end comment
00199
00200    /'
00201    A block
00202    of
00203    comment
00204    '/
00205    ~~~
00206    They're only visible in the FB source. The intermediate format
00207    contains empty lines in that case.
00208
00209
00210 \section sectExaDim Variables Declaration
00211
00212 A varible declaration in a FreeBasic source code is in human readable
00213 words (making it easy to understand the code) and may look like
00214
00215 ~~~{.bas}
00216 CONST VarName AS ZSTRING PTR
00217 ~~~
00218
00219 This couldn't get parsed by the C back-end, since this expects the
00220 short C syntax like
00221
00222 ~~~{.c}
00223 const char* VarName;
00224 ~~~
00225
00226 where
00227
00228 - `const` is a C keyword,
00229 - `char` is the name of an inbuild C type and the
00230 - `*` is similar to the FB keyword `PTR`.
00231
00232 The type declaration `char*` can get replaced by any other type. Rather
00233 than checking each type declaration, the parsers in the back-ends just
00234 scan the type names and use them to build the documentation.
00235
00236 fb-doc makes use of this flexibility. It creates new C types named
00237 similar to the FB syntax (without any declaration). As new type name
00238 all FB keywords get mangled in to a single word, separated by
00239 underscore characters. The C back-end interpretes this word as a type
00240 name and generates documentation output similar to the FB source code.
00241 The above example gets in the intermediate (C-like) format
00242
00243 ~~~{.c}
00244 CONST_ZSTRING_PTR VarName;
00245 ~~~
00246
00247 and will be used as *CONST_ZSTRING_PTR VarName* in the documentation as
00248 well. In this syntax the FB user can identify the type as an FB
00249 declaration and easily understand the documentation. See \ref
00250 sectTabInterForm for more examples.
00251
00252 \note The type name is allways in front of the symbol name, this order
00253       cannot get swapped as in FB syntax.
00254
00255 fb-doc can also generate real C types. In this case the above example
00256 gets
00257
00258 ~~~{.c}
00259 const char* VarName;
00260 ~~~
00261
00262 and this syntax can be used directly by a C compiler. This can be
```

```
00263 useful to auto-generate C headers from FB source, ie to bind a library
00264 written in FB in to a C project.
00265
00266
00267 \section sectExaTemplates Templates
00268
00269 In an early stage of the project development you'll start to add
00270 documentation comments to your source code. When targeting the Doxygen
00271 back-end this is an easy task. Just write the comment before or behind
00272 the relevant statement in the source code.
00273
00274 It's different for gtk-doc. Each and every documentation comment needs
00275 a list of the symbols the comment referes to. It's a boring job to copy
00276 the symbol names from the source code to the documentation comment.
00277 fb-doc can support this process.
00278
00279 \note The source code in this section contains verbatim blocks instead
00280       of code blocks, because Doxygen miss-interpretes the \\ \p brief
00281       as keyword (allthough it's inside the code block).
00282
00283
00284 \subsection subsectExaGtkdoc Gtk-doc
00285
00286 \note This example is based on a system wide installation of fb-doc
00287       and a setting as Geany custom command with option
00288       `--geany-mode`. (See \ref sectInstallGeany for details.)
00289
00290 This example is about documenting a function and its parameter list.
00291 We load our source code in to Geany IDE, select the function
00292 declaration (the code in the following box) and send it to the
00293 fb-doc custom command:
00294
00295 \verbatim
00296 FUNCTION goo_axis_new CDECL( _
00297   BYVAL Parent AS GooCanvasItem PTR, _
00298   BYVAL Back AS GooCanvasItem PTR, _
00299   BYVAL Text AS gchar PTR, _
00300   BYVAL Modus AS GooAxisType, _
00301   ...) AS GooAxis PTR
00302 \endverbatim
00303
00304 fb-doc extracts the names of the function and its parameters,
00305 generates a matching template and returns the customized template
00306 and the original code. The entries for the individual documentation
00307 texts are marked with the text `FIXME`:
00308
00309 \verbatim
00310 /'*
00311 goo_axis_new:
00312 @Parent: FIXME
00313 @Back: FIXME
00314 @Text: FIXME
00315 @Modus: FIXME
00316 @...: FIXME
00317
00318 FIXME
00319
00320 Returns: FIXME
00321 '/
00322 FUNCTION goo_axis_new CDECL( _
00323   BYVAL Parent AS GooCanvasItem PTR, _
00324   BYVAL Back AS GooCanvasItem PTR, _
00325   BYVAL Text AS gchar PTR, _
00326   BYVAL Modus AS GooAxisType, _
00327   ...) AS GooAxis PTR
00328 \endverbatim
00329
00330 Now we complete the documentation comment by filling the entries
00331 marked by the `FIXME`s, and we end-up with something like:
00332
00333 \verbatim
00334 /'*
00335 goo_axis_new:
00336 @Parent: the parent item, or %NULL. If a parent is specified, it
00337          will assume ownership of the item, and the item will
00338          automatically be freed when it is removed from the
00339          parent. Otherwise call g_object_unref() to free it.
00340   @Back: the background box to connect the axis to (a
00341          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
00342          Note: to set the axis position and size, the properties
00343          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
00344          #GooCanvasItemSimple:width and
00345          #GooCanvasItemSimple:height will be red (and therefore
00346          must be set in the background box item).
00347   @Text: the label text for the axis
00348  @Modus: the position and type (like %GOO_AXIS_SOUTH or
00349          %GOO_GRIDAXIS_SOUTH, ...)
```

```
00350    @...: optional pairs of property names and values, and a
00351          terminating %NULL.
00352
00353 Creates a new axis item.
00354
00355 Returns: (transfer full): a new axis item.
00356 '/
00357 FUNCTION goo_axis_new CDECL( _
00358   BYVAL Parent AS GooCanvasItem PTR, _
00359   BYVAL Back AS GooCanvasItem PTR, _
00360   BYVAL Text AS gchar PTR, _
00361   BYVAL Modus AS GooAxisType, _
00362   ...) AS GooAxis PTR
00363 \endverbatim
00364
00365 Finally we translate all FB sources to C source using (defaults: output
00366 path = `../doc/c_src`, file specification = `"*.bas" "*.bi"`)
00367
00368 ~~~{.sh}
00369 fbdoc --file-mode --asterix
00370 ~~~
00371
00372 The corresponding <em>.c</em> file looks like
00373
00374 \verbatim
00375 /**
00376 * goo_axis_new:
00377 * @Parent: the parent item, or %NULL. If a parent is specified, it
00378 *          will assume ownership of the item, and the item will
00379 *          automatically be freed when it is removed from the
00380 *          parent. Otherwise call g_object_unref() to free it.
00381 *   @Back: the background box to connect the axis to (a
00382 *          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
00383 *          Note: to set the axis position and size, the properties
00384 *          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
00385 *          #GooCanvasItemSimple:width and
00386 *          #GooCanvasItemSimple:height will be red (and therefore
00387 *          must be set in the background box item).
00388 *   @Text: the label text for the axis
00389 *  @Modus: the position and type (like %GOO_AXIS_SOUTH or
00390 *          %GOO_GRIDAXIS_SOUTH, ...)
00391 *    @...: optional pairs of property names and values, and a
00392 *          terminating %NULL.
00393 *
00394 * Creates a new axis item.
00395 *
00396 * Returns: (transfer full): a new axis item.
00397 * */
00398 FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (
00399 BYVAL_AS_GooCanvasItem_PTR Parent,
00400 BYVAL_AS_GooCanvasItem_PTR Back,
00401 BYVAL_AS_gchar_PTR Text,
00402 BYVAL_AS_GooAxisType Modus,
00403 ...) {
00404
00405 };
00406 \endverbatim
00407
00408 This C-like output can be used as input for the gtk-doc tool-chain to
00409 generate the desired documentation output. Execute the tools on the
00410 files in the <em>../doc/c_src</em> folder.
00411
00412 Or you use option `--cstyle` to generate output with types in real C syntax
00413
00414 ~~~{.sh}
00415 fbdoc --file-mode --asterix --cstyle
00416 ~~~
00417 and the corresponding <em>.c</em> file looks like
00418
00419 \verbatim
00420 /**
00421 * goo_axis_new:
00422 * @Parent: the parent item, or %NULL. If a parent is specified, it
00423 *          will assume ownership of the item, and the item will
00424 *          automatically be freed when it is removed from the
00425 *          parent. Otherwise call g_object_unref() to free it.
00426 *   @Back: the background box to connect the axis to (a
00427 *          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
00428 *          Note: to set the axis position and size, the properties
00429 *          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
00430 *          #GooCanvasItemSimple:width and
00431 *          #GooCanvasItemSimple:height will be red (and therefore
00432 *          must be set in the background box item).
00433 *   @Text: the label text for the axis
00434 *  @Modus: the position and type (like %GOO_AXIS_SOUTH or
00435 *          %GOO_GRIDAXIS_SOUTH, ...)
00436 *    @...: optional pairs of property names and values, and a
```

```
00437 *          terminating %NULL.
00438 *
00439 * Creates a new axis item.
00440 *
00441 * Returns: (transfer full): a new axis item.
00442 * */
00443 GooAxis* goo_axis_new (
00444 GooCanvasItem* Parent,
00445 GooCanvasItem* Back,
00446 gchar* Text,
00447 GooAxisType Modus,
00448 ...) {
00449
00450 };
00451 \endverbatim
00452
00453
00454 \subsection subsectExaDoxy Doxygen
00455
00456 \note This example is based on a system wide installation of fb-doc
00457       and a setting as Geany custom command with option `--geany-mode
00458       "DoxygenTemplates"`. (See \ref pageInstall for details.)
00459
00460 This example is about documenting a function and its parameter list,
00461 similar to the previous one. We load our code in to Geany IDE, select
00462 some source like the following code and send it to the fb-doc custom
00463 command:
00464
00465 \verbatim
00466 FUNCTION goo_axis_new CDECL( _
00467   BYVAL Parent AS GooCanvasItem PTR, _
00468   BYVAL Back AS GooCanvasItem PTR, _
00469   BYVAL Text AS gchar PTR, _
00470   BYVAL Modus AS GooAxisType, _
00471   ...) AS GooAxis PTR
00472 \endverbatim
00473
00474 fb-doc extracts the names and returns the customized template, followed
00475 by the original code:
00476
00477 \verbatim
00478 /'* \fn FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (BYVAL_AS_GooCanvasItem_PTR Parent,
       BYVAL_AS_GooCanvasItem_PTR Back, BYVAL_AS_gchar_PTR Text, BYVAL_AS_GooAxisType Modus, ...)
00479 \param Parent FIXME
00480 \param Back FIXME
00481 \param Text FIXME
00482 \param Modus FIXME
00483 \param ... FIXME
00484
00485 \brief FIXME
00486
00487 \returns FIXME
00488
00489 FIXME
00490
00491 '/
00492 FUNCTION goo_axis_new CDECL( _
00493   BYVAL Parent AS GooCanvasItem PTR, _
00494   BYVAL Back AS GooCanvasItem PTR, _
00495   BYVAL Text AS gchar PTR, _
00496   BYVAL Modus AS GooAxisType, _
00497   ...) AS GooAxis PTR
00498 \endverbatim
00499
00500 The first line is the declaration of our function in C syntax &mdash;
00501 without the character ; at the end and prepended by the keyword \\`fn`.
00502 This line enables Doxygen to read the documentation comment anywhere in
00503 its input, so the documentations comment needs not to be in front of
00504 the matching function. Since we don't wonna move this block (it's
00505 advantageous to place the documentation text near to the source code)
00506 we can remove this line and then complete the documentation comment by
00507 filling the `FIXME` entries. We end-up with something like:
00508
00509 \verbatim
00510 /'* \brief Creates a new axis item.
00511 \param Parent the parent item, or %NULL. If a parent is specified, it
00512          will assume ownership of the item, and the item will
00513          automatically be freed when it is removed from the
00514          parent. Otherwise call g_object_unref() to free it.
00515 \param Back the background box to connect the axis to (a
00516          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
00517          Note: to set the axis position and size, the properties
00518          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
00519          #GooCanvasItemSimple:width and
00520          #GooCanvasItemSimple:height will be red (and therefore
00521          must be set in the background box item).
00522 \param Text the label text for the axis
```

```
00523 \param Modus the position and type (like %GOO_AXIS_SOUTH or
00524          %GOO_GRIDAXIS_SOUTH, ...)
00525 \param ... optional pairs of property names and values, and a
00526          terminating %NULL.
00527
00528 \returns a new axis item.
00529
00530 A detailed desription of the function may follow here ...
00531
00532 '/
00533 FUNCTION goo_axis_new CDECL( _
00534   BYVAL Parent AS GooCanvasItem PTR, _
00535   BYVAL Back AS GooCanvasItem PTR, _
00536   BYVAL Text AS gchar PTR, _
00537   BYVAL Modus AS GooAxisType, _
00538   ...) AS GooAxis PTR
00539
00540 ' ...
00541 END FUNCTION
00542 \endverbatim
00543
00544 Using Doxygen we don't need to generate intermediate C source code in
00545 files. Instead this code gets directly piped by fb-doc acting as a
00546 filter, so we usually don't see it. Being curious we may execute in a
00547 terminal `fbdoc XYZ.bas` (replace XYZ by your file name) and we'll
00548 see output (which is the Doxygen input) like
00549
00550 \verbatim
00551 /**
00552 \param Parent the parent item, or %NULL. If a parent is specified, it
00553          will assume ownership of the item, and the item will
00554          automatically be freed when it is removed from the
00555          parent. Otherwise call g_object_unref() to free it.
00556 \param Back the background box to connect the axis to (a
00557          #GooCanvasRect, #GooCanvasImage, #GooCanvasGroup, ...).
00558          Note: to set the axis position and size, the properties
00559          #GooCanvasItemSimple:x, #GooCanvasItemSimple:y,
00560          #GooCanvasItemSimple:width and
00561          #GooCanvasItemSimple:height will be red (and therefore
00562          must be set in the background box item).
00563 \param Text the label text for the axis
00564 \param Modus the position and type (like %GOO_AXIS_SOUTH or
00565          %GOO_GRIDAXIS_SOUTH, ...)
00566 \param ... optional pairs of property names and values, and a
00567          terminating %NULL.
00568
00569 \brief Creates a new axis item.
00570
00571 \returns: a new axis item.
00572
00573 The detailed desription of the function follows here ...
00574
00575 */
00576 FUNCTION_CDECL_AS_GooAxis_PTR goo_axis_new (
00577 BYVAL_AS_GooCanvasItem_PTR Parent,
00578 BYVAL_AS_GooCanvasItem_PTR Back,
00579 BYVAL_AS_gchar_PTR Text,
00580 BYVAL_AS_GooAxisType Modus,
00581 ...) {
00582
00583 };
00584 \endverbatim
00585
00586 \note This output is also visible in the source code browser before the
00587       listings get repaired by executing `fbdoc -s` in the folder *doc*.
```

15.7 src/Extend.md File Reference

15.8 src/Extend.md

```
00001 Extending fb-doc  {#pageExtend}
00002 ================
00003 \tableofcontents
00004
00005 fb-doc is an open source project and everybody is able to customize the
00006 source code to his personal needs. Forking the project has the downside
00007 that changes have to be redone in case of an update of the original
00008 source.
00009
00010 fb-doc offers an alternative by its included interface for external
00011 emitters (= plugins). An external emitter is a shared library that gets
```

```
00012 loaded at runtime. The fb-doc project and the plugin are independend
00013 projects. In worst case the external modul has to get recompiled when
00014 the used headers were changed in fb-doc.
00015
00016 \section sectExtInternals Internals
00017
00018 Before we can extend fb-doc we should first understand some internals
00019 on how it works and how it serves the handlers in the \ref EmitterIF.
00020
00021 fb-doc loads the input in to a parsers buffer \ref Parser::Buf. Then
00022 the parser scans this context in two phases:
00023
00024 -# searching end of statements (new line or colon) and checking the
00025    start of the next statement
00026 -# if this is a relevant construct it gets tokenized by \ref
00027    Parser::tokenize()
00028
00029 So only relevant constructs get checked in detail. The tokenizer
00030 generates a list of tokens containing three INTEGER values for each
00031 entry:
00032
00033 -# the token type \ref Parser::ParserTokens
00034 -# the start of the token in the buffer (zero based) and
00035 -# its length in bytes
00036
00037 The complete construct gets tokenized. This is a single line in case of
00038 an \#`INCLUDE`, but also may be a bunch of lines in case of a block
00039 (\#`MACRO  ENUM  TYPE  SUB  ...`). The tokenizer uses just a subset of
00040 the FB keywords, check function \ref Parser::getToken() for details.
00041 All other words from the source code get `TOK_WORD` in the list. White
00042 spaces (like space, tab, caridge return, vertical tab, ...) and also
00043 other context without a specifier in \ref Parser::ParserTokens like
00044 numerical expressions get no entry in the token list. If this stuff is
00045 needed for an emitter, the emitter has to do the parsing itself.
00046
00047 Then the parser checks the token list at the beginning of the construct.
00048
00049 In case of syntax errors the handler \ref EmitterIF::Error_() gets
00050 called and the parser drops the construct. It's up to the emitter how
00051 and where to output the error message or to parse the buggy construct
00052 anyway.
00053
00054 In case of correct syntax the parser calls the matching emitter handler
00055 in the \ref EmitterIF. Priviously, during the syntax check, some
00056 pointers are set to the tokens in the list. The handler can use this
00057 pointers and the tokenlist to generate its output.
00058
00059 These most important pointers are set always
00060
00061 - \ref Parser::StaTok the first (start) token of the construct (*ie*
00062   `TOK_SUB` *in* `SUB name ...` *and* `TOK_DECL` *in* `DECLARE SUB
00063   name ...`)
00064 - \ref Parser::EndTok the last token in a construct (`TOK_EOS`, *check*
00065   \ref Parser::Buf[TokEnd[1]] <em>if it's a colon or a new line</em>)
00066 - \ref Parser::Tk The current position of the parser.
00067
00068 More than 15 further pointers are used in the parsers check, see \ref
00069 Parser for details. But not all pointers are set or reset for each
00070 construct. Find further information the source code of these parser
00071 functions:
00072
00073 |                 Function | Constructs                                                                |
00074 | ----------------------: | :------------------------------------------------------------------------ |
00075 | \ref Parser::FUNCTION_() | `SUB  FUNCTION  PROPERTY  OPERERATOR  CONSTRUTOR  DESTRUCTOR` |
00076 | \ref Parser::VAR_()      | `DIM  REDIM  VAR  CONST  COMMON  EXTERN  EXPORT  STATIC`      |
00077 | \ref Parser::TYPE_()     | `TYPE  CLASS`                                                             |
00078 | \ref Parser::UNION_()    | `UNION`                                                                   |
00079 | \ref Parser::ENUM_()     | `ENUM`                                                                    |
00080 | \ref Parser::DECLARE_()  | `DECLARE`                                                                 |
00081 | \ref Parser::DEFINE_()   | \#`DEFINE`                                                                |
00082 | \ref Parser::MACRO_()    | \#`MACRO`                                                                 |
00083 | \ref Parser::INCLUDE_()  | \#`INCLUDE`
      |
00084
00085 Some functions are available to parse advanced constructs like lists of
00086 several variable declarations:
00087
00088 - \ref Parser::parseListNam evaluate a list of names
00089 - \ref Parser::parseListNamTyp evaluate a list of declarations (name and type)
00090 - \ref Parser::parseListPara evaluate a parameter list
00091 - \ref Parser::parseBlockEnum
00092
00093 As paratmeter these helpers get a function pointer to an \ref EmitFunc
00094 handler. This handler gets called on each member of the list. Each
00095 member is of the same type here (parameter, enumerator, variable, ...).
00096
00097 It's a little more complex when you have to handle a `TYPE` or
```

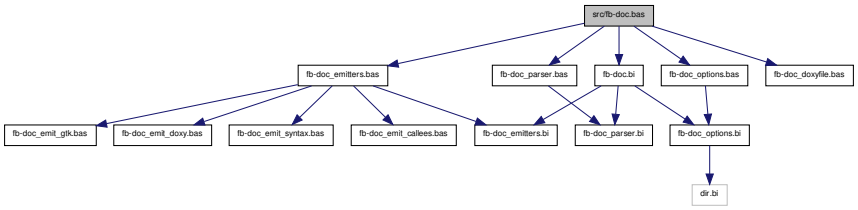
```
00098 'UNION' block. The helper function for this to parse is:
00099
00100 - \ref Parser::parseBlockTyUn
00101
00102 and the passed handler function has to deal with several kinds of
00103 context (like variable declarations or 'function declarations). Also
00104 these blocks may contain further nested blocks that may or may not get
00105 parsed recursivly (depending on the purpose of the emitter). The
00106 handler has to deal with all this kind of stuff. Find examples in any
00107 \ref EmitterIF::Clas_ handler, ie in the handler \ref c_Block().
00108
00109
00110 \section sectExtFlow Control Flow
00111
00112 When loading an external emitter plugin fb-doc first calls the function
00113 \ref EmitterInit to receive the pointer to the external \ref EmitterIF.
00114 This gets done right after parsing the command line and before a \ref
00115 Parser UDT is created.
00116
00117 ???
00118
00119
00120 \section sectExtPlugin Example Plugin.bas
00121
00122 Enough of theory, let's switch to practise. The fb-doc parser calls
00123 emitter functions via the emitter interface \ref EmitterIF, so it can
00124 easy get extended by new emitters providing additional features (ie
00125 support further C tools like <em>source navigator</em>).
00126
00127 The *src* folder of archive *fb-doc.zip* contains a file *Plugin.bas*.
00128 This file is an example source code for an external emitter modul and
00129 contains this further information:
00130
00131 \dontinclude Plugin.bas
\skipline This emitter generates
00132 \until endverbatim
00133
00134 \note The entry in \ref EmitterIF::Nam of an external emitter plugin is
00135       not used in fb-doc, its free for any usage.
00136
```

15.9 src/fb-doc.bas File Reference

The main source file to compile.

```
#include "fb-doc.bi"
#include "fb-doc_doxyfile.bas"
#include "fb-doc_parser.bas"
#include "fb-doc_emitters.bas"
#include "fb-doc_options.bas"
```

Include dependency graph for fb-doc.bas:



Functions

- int main ()

  A dummy function.

Variables

- const VAR PROG_NAME = "fb-doc"

  the name of the programm
- const VAR PROG_VERS = "0.3.9"

  the program version

15.9.1 Detailed Description

The main source file to compile. This main file includes all the submodules. Compile it by fbc (the FreeBASIC compiler) to create your fb-doc binary.

Definition in file fb-doc.bas.

15.9.2 Function Documentation

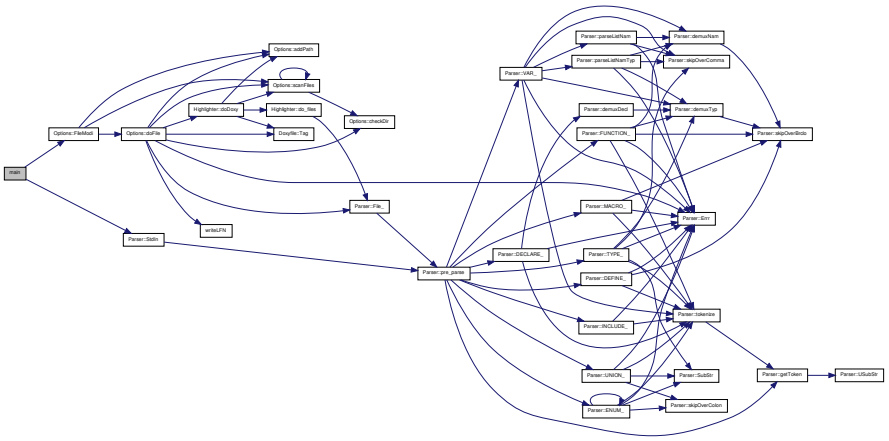
15.9.2.1 int main ( )

A dummy function.

This function is not in FB code. If it is omitted, then Doxygen doesn't create caller / callee graphs for the function calls in that code. That's why we cover the FB main part by a a peudo C function.

Definition at line 89 of file fb-doc.bas.

Here is the call graph for this function:



15.9.3 Variable Documentation

15.9.3.1 const VAR PROG_NAME = "fb-doc"

the name of the programm

Definition at line 3 of file fb-doc.bas.

15.9.3.2 const VAR PROG_VERS = "0.3.9"

the program version

Definition at line 4 of file fb-doc.bas.

15.10 fb-doc.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 ' coding: UTF-8
00002 ' This is
00003 CONST PROG_NAME = "fb-doc" '*< the name of the programm
00004 CONST PROG_VERS = "0.3.9"  '*< the program version
00005
00006 /'* \file fb-doc.bas
00007 \brief The main source file to compile
00008
00009 This main file includes all the submodules. Compile it by
00010 fbc (the FreeBASIC compiler) to create your fb-doc binary.
00011
00012 '/
00013
00014
00015 /'* \mainpage fb-doc
00016
00017 fb-doc is a tool for generating documentation from and for FreeBasic
00018 source code. Rather than creating the documentation output directly,
00019 fb-doc is designed to close the gap between FreeBasic (= FB) and C
00020 syntax to allow for using any C / C++ documentation tool-chain. Also
00021 it supports the creation of documentational comments in the source
00022 and it can get extended via external plugins.
00023
00024 Find more information in the tutorial pages
00025
00026 - \subpage pageIntro
00027 - \subpage pageInstall
00028 - \subpage pageUsage
00029 - \subpage pageTables
00030 - \subpage pageOptionDetails
00031 - \subpage pageEmitterDetails
00032 - \subpage pageFilesUsed
00033 - \subpage pageExamples
00034 - \subpage pageExtend
00035 - \subpage pageToDo
00036
00037 or at the world wide web:
00038
00039  - en: \ProjPage
00040  - de: http://www.freebasic-portal.de/downloads/ressourcencompiler/fb-doc-229.html
00041
00042
00043 \section licence fb-doc licence (GPLv3):
00044
00045 Copyright &copy; 2012-2014 by Thomas{ DoT ]Freiherr[ aT ]gmx[ dOt }net
00046
00047 This program is free software; you can redistribute it and/or modify
00048 it under the terms of the GNU General Public License as published by
00049 the Free Software Foundation; either version 2 of the License, or (at
00050 your option) any later version.
00051
00052 This program is distributed in the hope that it will be useful, but
00053 WITHOUT ANY WARRANTY; without even the implied warranty of
00054 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00055 General Public License for more details.
00056
00057 You should have received a copy of the GNU General Public License
00058 along with this program; if not, write to the Free Software
00059 Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-
00060 1301, USA. For further details please refer to:
00061 http://www.gnu.org/licenses/gpl-3.0.html
00062
00063
00064 \section ackno Acknowledgements
00065
00066 Thanks go to:
00067
00068  - Dimitri van Heesch (author of Doxygen)
00069  - Chris Lyttle, Dan Mueth, Stefan Kost (authors of gtk-doc)
00070  - the FreeBasic developers (a great modern programming language with fun-factor)
00071  - AGS (from http://www.freebasic.net/forum) for testing and bug reporting
00072
00073 '/
00074
00075 #INCLUDE ONCE "fb-doc.bi"
00076
00077 #INCLUDE ONCE "fb-doc_doxyfile.bas"
00078 #INCLUDE ONCE "fb-doc_parser.bas"
00079 #INCLUDE ONCE "fb-doc_emitters.bas"
00080 #INCLUDE ONCE "fb-doc_options.bas"
00081
00082 /'* \brief A dummy function
00083
```

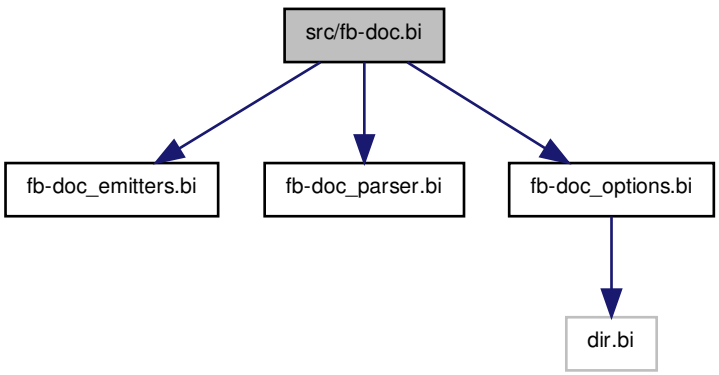
```
00084 This function is not in FB code. If it is omitted, then Doxygen doesn't
00085 create caller / callee graphs for the function calls in that code.
00086 That's why we cover the FB main part by a a peudo C function.
00087
00088 '/
00089 '&int main () { /* dummy function for Doxygen */
00090
00091 OPT = NEW Options()
00092
00093 WITH *.OPT
00094   IF .Efnr THEN
00095     SELECT CASE AS CONST .parseCLI()
00096     CASE   .ERROR_MESSAGE : ERROUT("Invalid command line (" & MID(.Errr, 3) & ")")
00097     CASE    .HELP_MESSAGE : ERROUT(MSG_WELCOME & MSG_HELP)
00098     CASE .VERSION_MESSAGE : ERROUT(MSG_WELCOME & MSG_VERSION)
00099     CASE .GEANY_MODE
00100       .InTree = 0 '                                    ignore user setting
00101       .Ocha = FREEFILE
00102       OPEN CONS FOR OUTPUT AS #.Ocha
00103       .Pars->StdIn()
00104 '&    Parser.StdIn() /* dummy call for Doxygen */
00105       CLOSE #.Ocha
00106     CASE ELSE
00107       .FileModi()
00108 '&    Options.FileModi() /* dummy call for Doxygen */
00109     END SELECT
00110   END IF
00111 END WITH
00112
00113 DELETE OPT
00114 FOR i AS INTEGER = 0 TO UBOUND(Emitters)
00115   DELETE Emitters(i)
00116 NEXT
00117
00118 '&}
00119
```

15.11 src/fb-doc.bi File Reference

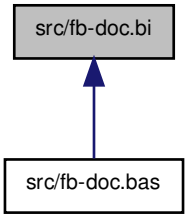
Header for the main source file.

```
#include "fb-doc_emitters.bi"
#include "fb-doc_parser.bi"
#include "fb-doc_options.bi"
```

Include dependency graph for fb-doc.bi:



This graph shows which files directly or indirectly include this file:



Macros

- #define MSG_HELP

  The help message for the command line interface (option `--help`)
- #define MSG_WELCOME

  The welcome message, shown when running as a program (option `--file-mode`)
- #define MSG_VERSION

  The version information for the command line interface (option `--version`)
- #define FBDOC_TARGET /∗ "UNIX/LINUX" ∗/

  The target operation system (used in MSG_VERSION, UNIX version here)
- #define FBDOC_BINARY /∗ "./" & PROG_NAME ∗/

  The name of the binary (used in MSG_HELP, UNIX version here)
- #define FBDOC_BINARY /∗ PROG_NAME & ".exe" ∗/

  The name of the binary (used in MSG_HELP, UNIX version here)

Variables

- ∗const VAR COMM_END = NL & NL & "'/" & NL

  the end of an FB comment block (in templates)
- ∗const VAR FIXME = "FIXME"

  the text to initialize entry fields (in templates)

15.11.1 Detailed Description

Header for the main source file. This file contains #`DEFINE`s for the user communication at the command prompt and adjusts some operation system specific settings. It also includes the headers for the modules.

Definition in file fb-doc.bi.

15.11.2 Macro Definition Documentation

15.11.2.1 #define MSG_HELP

Value:

```
/*  _
  "Command line tool for generating documentation from FreeBasic source code." & NL & NL & _
  "Usage" & NL & _
  "   " & FBDOC_BINARY & " [Options]" & NL & NL & _
  "Options:" & NL & _
  " -h (--help)    : print this help text and stop" & NL & _
  " -v (--version) : print version information and stop" & NL & _
  "       none        : file in --> STDOUT" & NL & _
  " -f (--file-mode)  : file in --> file out" & NL & _
  " -g (--geany-mode) : STDIN --> STDOUT" & NL & _
  " -l (--list-mode)  : Doxgfile inputs --> fb-doc.lfn" & NL & _
  " -s (--syntax-mode): scan doxygen output, repair syntax highlighting" & NL & _
  " -a (--asterix) : prepend '* ' in ML comments (gtk-doc style)" & NL & _
  " -c (--cstyle)  : emit real C types" & NL & _
  " -e (--emitter) : specify emitter name" & NL & _
  " -o (--outpath) : specify output directory" & NL & _
  " -r (--recursiv): scan input files also in subfolders" & NL & _
  " -t (--tree)    : scan source tree (follow #INCLUDEs)" & NL & _
  "Example:" & NL & _
  "   " & FBDOC_BINARY & " --geany-mode" & NL & _
  "      Get input from STDIN, prepend a matching comment block, emit to STDOUT" & NL & _
  "      (emits gtk-doc templates for ENUM, UNION, TYPE, SUB, FUNCTION, PROPERTY)" & NL & _
  "   " & FBDOC_BINARY & " -f -t MyProject.bas" & NL & _
  "      Load MyProject.bas in current folder and follow source tree" & NL & _
  "      emit pseudo C code in ../doc/c_src" & NL & _
  "For details see file 'ReadMe.txt' or visit:" & NL & _
  "  http://www.freebasic-portal.de/downloads/ressourcencompiler/fb-doc-229.html (de)" & NL & _
  "  http://www.freebasic.net/forum/viewtopic.php?f=8&t=19810 (en)" & NL */
```

The help message for the command line interface (option `--help`)

Definition at line 11 of file fb-doc.bi.

15.11.2.2 #define MSG_WELCOME

Value:

```
/*  _
  "version " & PROG_VERS & ", License GPLv3" & NL & _
  "        Copyright (C) 2012-2014 by Thomas{ At ]Freiherr{ at }gmx[ dOt ]net" & NL */
```

The welcome message, shown when running as a program (option `--file-mode`)

Definition at line 41 of file fb-doc.bi.

15.11.2.3 #define MSG_VERSION

Value:

```
/*  _
  "  Compiled: " & __DATE__ & ", " & __TIME__ & " for " & _
  FBDOC_TARGET & ". (" & __FB_SIGNATURE__ & ")" & NL */
```

The version information for the command line interface (option `--version`)

Definition at line 46 of file fb-doc.bi.

15.11.2.4 #define FBDOC_TARGET /* "UNIX/LINUX" */

The target operation system (used in MSG_VERSION, UNIX version here)

Definition at line 52 of file fb-doc.bi.

15.11.2.5 #define FBDOC_BINARY /* "./" & PROG_NAME */

The name of the binary (used in MSG_HELP, UNIX version here)

The operation system specific slash character (UNIX version here)

Definition at line 59 of file fb-doc.bi.

15.11.2.6 #define FBDOC_BINARY /* PROG_NAME & ".exe" */

The name of the binary (used in MSG_HELP, UNIX version here)

The operation system specific slash character (UNIX version here)

Definition at line 59 of file fb-doc.bi.

15.11.3 Variable Documentation

15.11.3.1 * const VAR COMM_END = NL & NL & "'/" & NL

the end of an FB comment block (in templates)

Definition at line 68 of file fb-doc.bi.

15.11.3.2 * const VAR FIXME = "FIXME"

the text to initialize entry fields (in templates)

Definition at line 69 of file fb-doc.bi.

15.12 fb-doc.bi

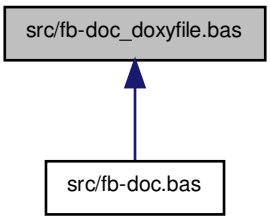
```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc.bi
00002 \brief Header for the main source file
00003
00004 This file contains \#'DEFINE's for the user communication at the
00005 command prompt and adjusts some operation system specific settings.
00006 It also includes the headers for the modules.
00007
00008 '/
00009
00010 /'* \brief The help message for the command line interface (option '--help') '/
00011 #DEFINE MSG_HELP _
00012   "Command line tool for generating documentation from FreeBasic source code." & NL & NL & _
00013   "Usage" & NL & _
00014   "   " & FBDOC_BINARY & " [Options]" & NL & NL & _
00015   "Options:" & NL & _
00016   " -h (--help)    : print this help text and stop" & NL & _
00017   " -v (--version) : print version information and stop" & NL & _
00018   "       none         : file in --> STDOUT" & NL & _
00019   " -f (--file-mode)  : file in --> file out" & NL & _
00020   " -g (--geany-mode) : STDIN --> STDOUT" & NL & _
00021   " -l (--list-mode)  : Doxgfile inputs --> fb-doc.lfn" & NL & _
00022   " -s (--syntax-mode): scan doxygen output, repair syntax highlighting" & NL & _
00023   " -a (--asterix) : prepend '* ' in ML comments (gtk-doc style)" & NL & _
00024   " -c (--cstyle)  : emit real C types" & NL & _
00025   " -e (--emitter) : specify emitter name" & NL & _
00026   " -o (--outpath) : specify output directory" & NL & _
00027   " -r (--recursiv): scan input files also in subfolders" & NL & _
00028   " -t (--tree)    : scan source tree (follow #INCLUDEs)" & NL & _
00029   "Example:" & NL & _
00030   "   " & FBDOC_BINARY & " --geany-mode" & NL & _
00031   "      Get input from STDIN, prepend a matching comment block, emit to STDOUT" & NL & _
00032   "      (emits gtk-doc templates for ENUM, UNION, TYPE, SUB, FUNCTION, PROPERTY)" & NL & _
00033   "   " & FBDOC_BINARY & " -f -t MyProject.bas" & NL & _
00034   "      Load MyProject.bas in current folder and follow source tree" & NL & _
00035   "      emit pseudo C code in ../doc/c_src" & NL & _
00036   "For details see file 'ReadMe.txt' or visit:" & NL & _
00037   "  http://www.freebasic-portal.de/downloads/ressourcencompiler/fb-doc-229.html (de)" & NL & _
00038   "  http://www.freebasic.net/forum/viewtopic.php?f=8&t=19810 (en)" & NL
00039
00040 /'* \brief The welcome message, shown when running as a program (option '--file-mode') '/
00041 #DEFINE MSG_WELCOME _
00042   "version " & PROG_VERS & ", License GPLv3" & NL & _
00043   "        Copyright (C) 2012-2014 by Thomas{ At ]Freiherr{ at }gmx[ dOt ]net" & NL
00044
00045 /'* \brief The version information for the command line interface (option '--version') '/
00046 #DEFINE MSG_VERSION _
00047   "  Compiled: " & __DATE__ & ", " & __TIME__ & " for " & _
00048   FBDOC_TARGET & ". (" & __FB_SIGNATURE__ & ")" & NL
```

```
00049
00050 #IFDEF __FB_UNIX__
00051 /'* \brief The target operation system (used in \ref MSG_VERSION, UNIX version here) '/
00052  #DEFINE FBDOC_TARGET "UNIX/LINUX"
00053 /'* \brief The name of the binary (used in \ref MSG_HELP, UNIX version here) '/
00054  #DEFINE FBDOC_BINARY "./" & PROG_NAME
00055 /'* \brief The operation system specific slash character (UNIX version here) '/
00056 #ELSE
00057 '&/*
00058  #DEFINE FBDOC_TARGET "win/dos"
00059  #DEFINE FBDOC_BINARY PROG_NAME & ".exe"
00060 '&*/
00061 #ENDIF
00062
00063 #INCLUDE ONCE "fb-doc_emitters.bi"
00064 #INCLUDE ONCE "fb-doc_parser.bi"
00065 #INCLUDE ONCE "fb-doc_options.bi"
00066
00067 CONST _
00068   COMM_END = NL & NL & "'/" & NL, _ '*< the end of an FB comment block (in templates)
00069      FIXME = "FIXME"                '*< the text to initialize entry fields (in templates)
00070
```

15.13 src/fb-doc_doxyfile.bas File Reference

The source code for the Doxyfile class.

This graph shows which files directly or indirectly include this file:



Classes

- class Doxyfile

  handle a Doxyfile

15.13.1 Detailed Description

The source code for the Doxyfile class. This file contains the source code for a class used to read parameters from a Doxyfile, a file that contain settings to control the operations of Doxygen. fb-doc reads this files to operate on the same folders and files as Doxygen.

This is used in modes `--list-mode` and `--syntax-mode`.

Definition in file fb-doc_doxyfile.bas.

15.14 fb-doc_doxyfile.bas

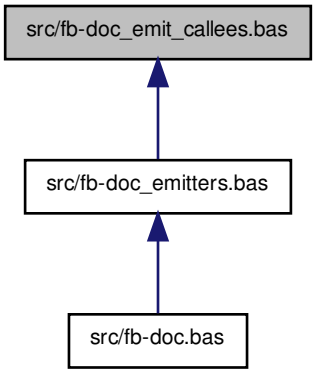
```
%%% Syntax-highlighting by fb-doc %%%
```

```
00001 /'* \file fb-doc_doxyfile.bas
00002 \brief The source code for the \ref Doxyfile class
00003
00004 This file contains the source code for a class used to read parameters
00005 from a Doxyfile, a file that contain settings to control the operations
00006 of Doxygen. fb-doc reads this files to operate on the same folders and
00007 files as Doxygen.
00008
00009 This is used in modes `--list-mode` and `--syntax-mode`.
00010
00011 '/
00012
00013 /'* \brief handle a Doxyfile
00014
00015 Read a Doxyfile and parse its coontext. In case of errors finding or
00016 loading the file the Errr variable gets set in the constructor. Its
00017 empty on success. Currently only single line tags are supported (no
00018 lists).
00019
00020 '/
00021 TYPE Doxyfile
00022   AS INTEGER Length
00023   AS UBYTE PTR Buffer
00024   AS ZSTRING PTR Doxy = @"empty"
00025   AS STRING Errr
00026
00027   DECLARE CONSTRUCTOR(BYREF AS STRING)
00028   DECLARE DESTRUCTOR()
00029   DECLARE FUNCTION Tag(BYREF AS STRING) AS STRING
00030 END TYPE
00031
00032
00033 /'* \brief construct reading a Doxyfile
00034 \param Fnam The (path and) file name
00035
00036 The constructor tries to find and read the Doxyfile. In case of
00037 errors the variable Errr contains an error message. Its empty
00038 otherwise and the context can get parsed by the function \ref Tag().
00039
00040 '/
00041 CONSTRUCTOR Doxyfile(BYREF Fnam AS STRING)
00042   VAR fnr = FREEFILE
00043   IF OPEN(Fnam FOR INPUT AS fnr) THEN errr = "error (couldn't open)" : EXIT CONSTRUCTOR
00044
00045   Length = LOF(fnr)
00046   Buffer = ALLOCATE(Length + 1)
00047   IF 0 = Buffer THEN errr = "error (out of memory)" : CLOSE #fnr : Length = 0 : EXIT CONSTRUCTOR
00048
00049   VAR bytes_red = 0
00050   GET #fnr, 1, *Buffer, Length, bytes_red
00051   IF Length <> bytes_red THEN _
00052     errr = "read error (red " & bytes_red & " of " & Length & " bytes)" : Length = 0 : EXIT CONSTRUCTOR
00053   Buffer[bytes_red] = 0
00054   Doxy = CAST(ZSTRING PTR, Buffer)
00055   CLOSE #fnr
00056 END CONSTRUCTOR
00057
00058 /'* \brief destructor freeing the memory used
00059
00060 The destructor frees the memory allocated in the constructor, if any.
00061
00062 '/
00063 DESTRUCTOR Doxyfile()
00064   IF Buffer THEN DEALLOCATE(Buffer)
00065 END DESTRUCTOR
00066
00067 /'* \brief find a single line tag in the Doxyfile and return its value
00068 \param Su the tag to search for
00069 \returns the tags value (if any))
00070
00071 This function is designed to find a value of an Doxyfile tag. It
00072 return the tags context or an empty string if the tag has no value
00073 or cannot be found.
00074
00075 '/
00076 FUNCTION Doxyfile.Tag(BYREF Su AS STRING) AS STRING
00077   VAR p1 = INSTR(*Doxy, !"\n" & Su & " ") : IF 0 = p1 THEN RETURN ""
00078   VAR p2 = INSTR(p1 + LEN(Su), *Doxy, "="), _
00079       p3 = INSTR(p2 + 1, *Doxy, !"\n") - 1
00080   RETURN TRIM(MID(*Doxy, p2 + 1, p3 - p2))
00081 END FUNCTION
00082
00083
```

15.15 src/fb-doc_emit_callees.bas File Reference

Emitter for the file *fb-doc.lfn*.

This graph shows which files directly or indirectly include this file:



Functions

- FUNCTION_AS_INTEGER writeLFN (BYref_AS_STRING Path)
- SUB_CDECL callees_decl_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a declaration line.
- SUB_CDECL callees_class_ (BYVAL_AS_Parser_PTR P)

  Emitter to start parsing of blocks.
- SUB_CDECL callees_func_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a line for a function name.
- SUB_CDECL callees_include (BYVAL_AS_Parser_PTR P)

  Emitter to import a source file.

Variables

- const VAR CALLEE_TR = !"\n"

  Separator for entries in file fb-doc.lfn.

15.15.1 Detailed Description

Emitter for the file *fb-doc.lfn*. This file contains emitter functions for the EmitterIF to generate the file *fb-doc.lfn* for the Doxygen Back-end. It's the default emitter in mode `--list-mode`.

The emitters writes the names of all functions (SUB / FUNCTION / PROPERTY) to the output stream, one in a line, separated by a new line character "CHR(10)".

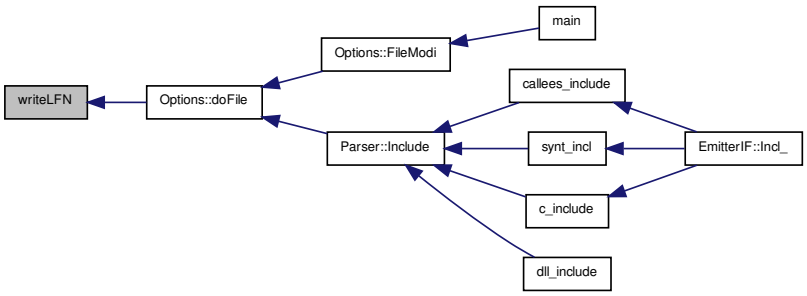
Definition in file fb-doc_emit_callees.bas.

15.15.2 Function Documentation

15.15.2.1 FUNCTION_AS_INTEGER writeLFN (BYref_AS_STRING *Path*)

Definition at line 17 of file fb-doc_emit_callees.bas.

Here is the caller graph for this function:



15.15.2.2 SUB_CDECL callees_decl_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a declaration line.

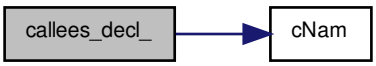
Parameters

P	the parser calling this emitter

This emitter gets called when the parser is in a declaration (VAR / DIM / CONST / COMMON / EXTERN / STATIC / DECLARE). It generates a line for each variable name and sends it (them) to the output stream.

Definition at line 32 of file fb-doc_emit_callees.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.15.2.3 SUB_CDECL callees_class_ (BYVAL_AS_Parser_PTR *P*)

Emitter to start parsing of blocks.

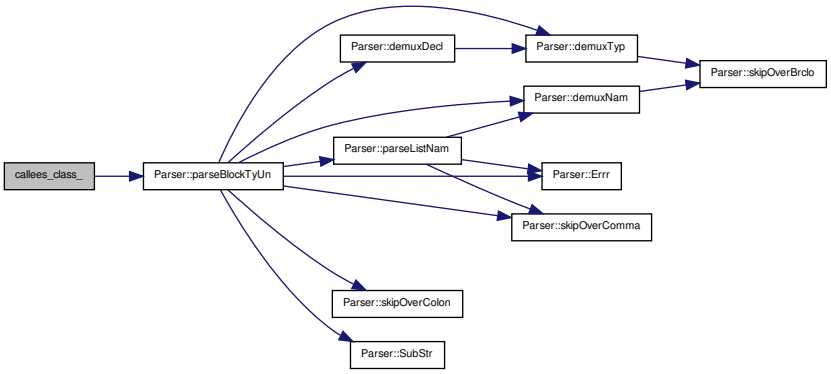
Parameters

P	the parser calling this emitter

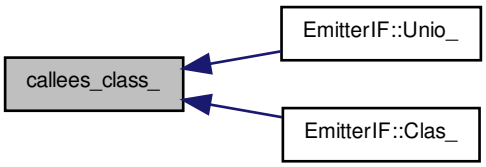
This emitter gets called when the parser finds a block (`TYPE UNION ENUM`). It starts the scanning process in the block.

Definition at line 52 of file fb-doc_emit_callees.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.15.2.4 SUB_CDECL callees_func_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a line for a function name.

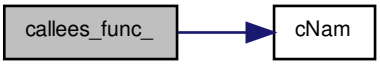
Parameters

P	the parser calling this emitter

This emitter gets called when the parser finds a function (`SUB FUNCTION PROPERTY`). It generates a line with the name of the function and sends it to the output stream.

Definition at line 67 of file fb-doc_emit_callees.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.15.2.5 SUB_CDECL callees_include (BYVAL_AS_Parser_PTR *P*)

Emitter to import a source file.

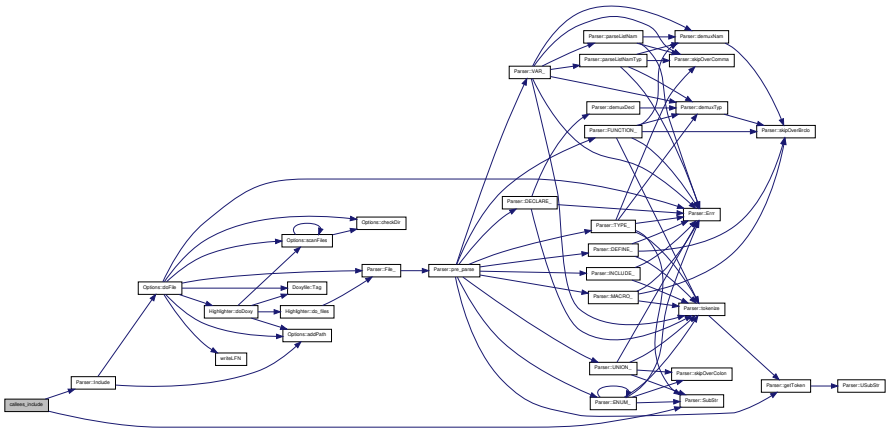
Parameters

P	the parser calling this emitter

This emitter gets called when the parser finds an #INCLUDE statement and option --recursiv is given. It checks if the file has been done already. If not, it creates a new Parser and starts the scanning process.

Definition at line 88 of file fb-doc_emit_callees.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.15.3 Variable Documentation

15.15.3.1 const VAR CALLEE_TR = !"\n"

Separator for entries in file *fb-doc.lfn*.

Definition at line 14 of file fb-doc_emit_callees.bas.

15.16 fb-doc_emit_callees.bas

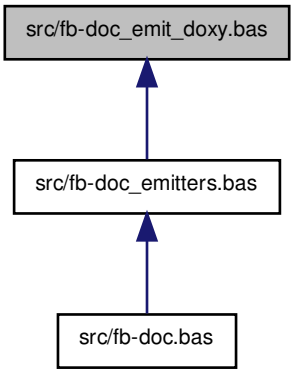
```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emit_callees.bas
00002 \brief Emitter for the file \em fb-doc.lfn.
00003
00004 This file contains emitter functions for the \ref EmitterIF to
00005 generate the file \em fb-doc.lfn for the Doxygen Back-end. It's the
00006 default emitter in mode `--list-mode`.
00007
00008 The emitters writes the names of all functions (SUB / FUNCTION /
00009 PROPERTY) to the output stream, one in a line, separated by a new
00010 line character "CHR(10)".
00011
00012 '/
00013
00014 CONST CALLEE_TR = !"\n" '*< Separator for entries in file \em fb-doc.lfn.
00015
00016
00017 FUNCTION writeLFN(BYref Path AS STRING) AS INTEGER
00018   var fnr = FREEFILE
00019   IF OPEN(Path & CALLEES_FILE FOR OUTPUT AS #fnr) THEN RETURN 0
00020   PRINT #fnr, "+++ List of Function Names +++"
00021   RETURN fnr
00022 END FUNCTION
00023
00024 /'* \brief Emitter to generate a declaration line
00025 \param P the parser calling this emitter
00026
00027 This emitter gets called when the parser is in a declaration (VAR /
00028 DIM / CONST / COMMON / EXTERN / STATIC / DECLARE). It generates a line for
00029 each variable name and sends it (them) to the output stream.
00030
00031 '/
00032 SUB callees_decl_ CDECL(BYVAL P AS Parser PTR)
00033   WITH *P
00034     SELECT CASE AS CONST *.StaTok
00035     CASE .TOK_SUB, .TOK_FUNC, .TOK_PROP
00036     CASE ELSE : EXIT SUB
00037     END SELECT : IF 0 = .NamTok ORELSE 0 = .FunTok THEN EXIT SUB
00038
00039     cNam(P)
00040     Code(CALLEE_TR)
00041   END WITH
00042 END SUB
00043
00044
00045 /'* \brief Emitter to start parsing of blocks
00046 \param P the parser calling this emitter
00047
00048 This emitter gets called when the parser finds a block (`TYPE  UNION
00049 ENUM`). It starts the scanning process in the block.
00050
```

```
00051 ''/
00052 SUB callees_class_ CDECL(BYVAL P AS Parser PTR)
00053   WITH *P
00054     IF OPT->AllCallees THEN .parseBlockTyUn(@callees_decl_)
00055   END WITH
00056 END SUB
00057
00058
00059 /''* \brief Emitter to generate a line for a function name
00060 \param P the parser calling this emitter
00061
00062 This emitter gets called when the parser finds a function (`SUB
00063 FUNCTION  PROPERTY`). It generates a line with the name of the
00064 function and sends it to the output stream.
00065
00066 ''/
00067 SUB callees_func_ CDECL(BYVAL P AS Parser PTR) '' !!! ToDo member functions
00068   WITH *P
00069     SELECT CASE AS CONST *.StaTok
00070     CASE .TOK_SUB, .TOK_FUNC, .TOK_PROP
00071     CASE ELSE : EXIT SUB
00072     END SELECT
00073     cNam(P)
00074     Code(CALLEE_TR)
00075   END WITH
00076 END SUB
00077
00078
00079 /''* \brief Emitter to import a source file
00080 \param P the parser calling this emitter
00081
00082 This emitter gets called when the parser finds an \#`INCLUDE`
00083 statement and option `--recursiv` is given. It checks if the file
00084 has been done already. If not, it creates a new #Parser and starts
00085 the scanning process.
00086
00087 ''/
00088 SUB callees_include CDECL(BYVAL P AS Parser PTR)
00089   WITH *P
00090     IF OPT->InTree THEN .Include(TRIM(.SubStr(.NamTok), """"))
00091   END WITH
00092 END SUB
00093
00094
00095 WITH_NEW_EMITTER("FunctionNames")
00096   .Clas_ = @callees_class_
00097   .Unio_ = @callees_class_
00098   .Func_ = @callees_func_
00099   .Decl_ = @callees_decl_
00100   .Incl_ = @callees_include
00101 END WITH
00102
00103
```

15.17 src/fb-doc_emit_doxy.bas File Reference

Emitter for Doxygen templates.

This graph shows which files directly or indirectly include this file:



Functions

- SUB_CDECL doxy_entryListPara (BYVAL_AS_Parser_PTR P)

  Emitter to generate a line for a parameter list entry.
- SUB_CDECL doxy_func_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a template for a function.
- SUB_CDECL doxy_decl_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a template for a declaration.
- SUB_CDECL doxy_defi_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a templates for a macro.
- SUB_CDECL doxy_emitBlockNames (BYVAL_AS_Parser_PTR P)

  Emitter to generate a line for a block entry.
- SUB_CDECL doxy_Block (BYVAL_AS_Parser_PTR P)

  Emitter to generate templates for blocks.
- SUB_CDECL doxy_empty (BYVAL_AS_Parser_PTR P)

  Emitter for an empty Geany block.

Variables

- const VAR DOXY_START = NL & "/'* "

  the start of a comment block
- const VAR DOXY_END

  the end of a comment block

15.17.1 Detailed Description

Emitter for Doxygen templates. This file contains emitter functions for the EmitterIF to generate template blocks for the Doxygen Back-end. It's designed to be used in Geany mode (see section Doxygen for an example).

The emitters return all original source code unchanged. Additionally relevant lines (or code blocks) get prepended by a multi line block of documentation. This works for `SUB`s, `FUNCTION`s, `TYPE, UNION` and `ENUM` blocks, declarations (`VAR DIM CONST COMMON EXTERN STATIC`) and #`DEFINE`s or #`MACRO`s.

The block starts with the C declaration of the construct and a line for the brief description. Then the members are listed with leading param or var keyword and followed by a line for the brief description. The block ends by a `FIXME` text for the detailed description.

Since the prefered way to document variables with Doxygen is to write the comment behind the variable, this emitter is mostly helpful for documenting functions and their parameter lists.

When an empty line is send by Geany, the text of the doxy_empty() function gets emitted. This is a comment block template to describe the source file.

Definition in file fb-doc_emit_doxy.bas.

15.17.2 Function Documentation

15.17.2.1 SUB_CDECL doxy_entryListPara (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a line for a parameter list entry.

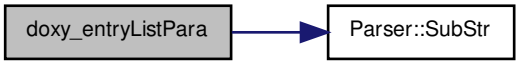
Parameters

P	the parser calling this emitter

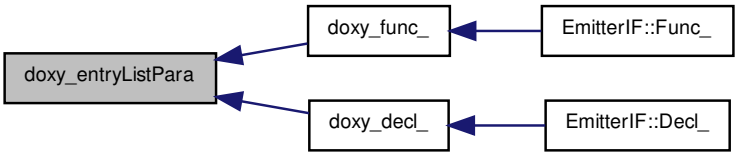
This emitter gets called when the parser is in a parameter list of a function (`SUB FUNCTION PROPERTY CONSTRUCTOR DESTRUCTOR`). It generates a line for each parameter and sends it (them) to the output stream.

Definition at line 47 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.2 SUB_CDECL doxy_func_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a template for a function.

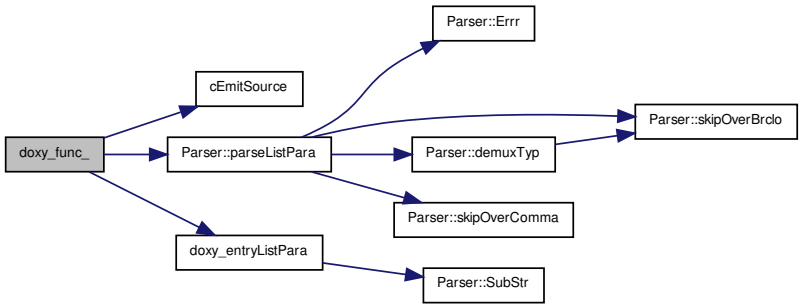
Parameters

P	the parser calling this emitter

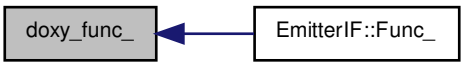
This emitter gets called when the parser finds a function (`SUB FUNCTION PROPERTY CONSTRUCTOR DESTRUCTOR`). It generates a template to document the function and its parameter list and sends it to the output stream.

Definition at line 63 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.3 SUB_CDECL doxy_decl_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a template for a declaration.

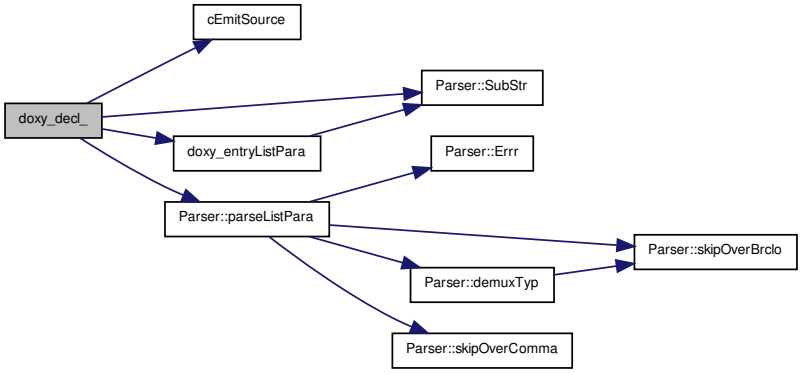
Parameters

P	the parser calling this emitter

This emitter gets called when the parser is in a declaration (`VAR DIM CONST COMMON EXTERN STATIC`). It generates a line for each variable name and sends it (them) to the output stream.

Definition at line 87 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.17.2.4 SUB_CDECL doxy_defi_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a templates for a macro.

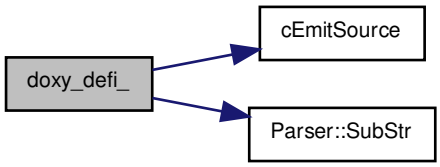
Parameters

P	the parser calling this emitter

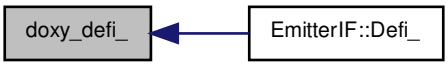
This emitter gets called when the parser finds a macro (#DEFINE #MACRO). It generates a template to document the macro and sends it to the output stream.

Definition at line 137 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.5 SUB_CDECL doxy_emitBlockNames (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a line for a block entry.

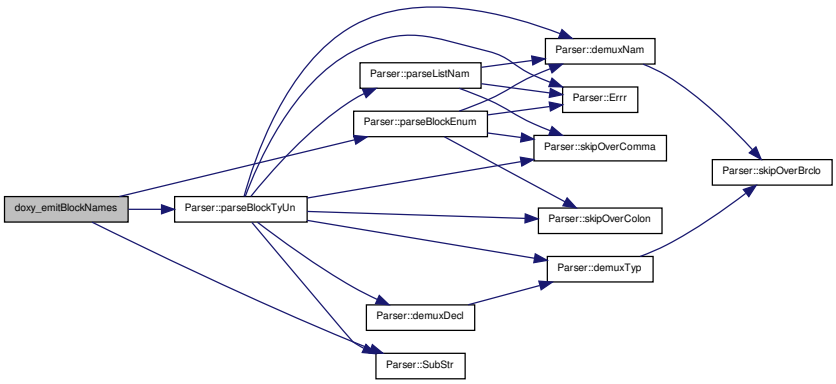
Parameters

P	the parser calling this emitter

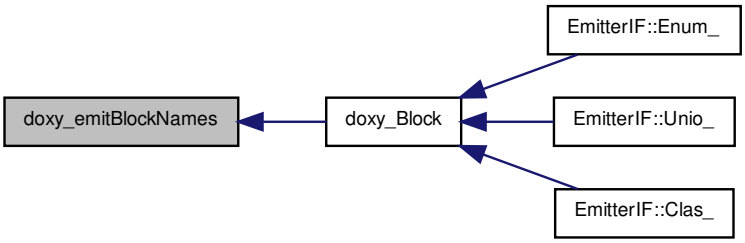
This emitter gets called when the parser is in a block (`TYPE ENUM UNION`). It generates a line for each member and sends it (them) to the output stream.

Definition at line 155 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.6 SUB_CDECL doxy_Block (BYVAL_AS_Parser_PTR *P*)

Emitter to generate templates for blocks.

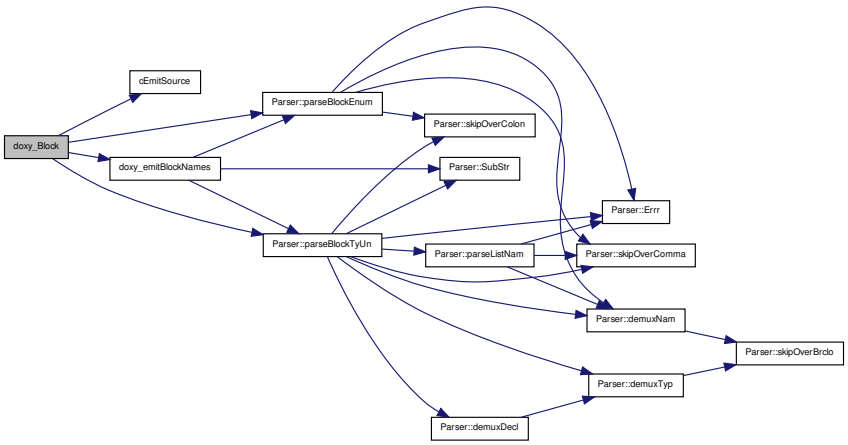
Parameters

P	the parser calling this emitter

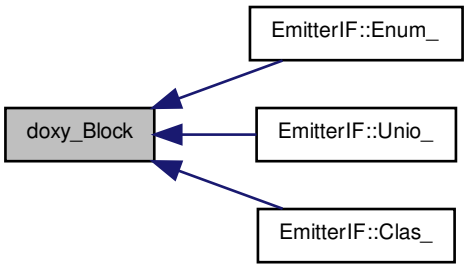
This emitter gets called when the parser finds a block (`TYPE UNION ENUM`). It generates a template to document the block with one line for each member and sends it to the output stream.

Definition at line 181 of file fb-doc_emit_doxy.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.7 SUB_CDECL doxy_empty (BYVAL_AS_Parser_PTR *P*)

Emitter for an empty Geany block.

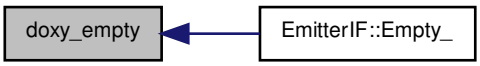
Parameters

P	the parser calling this emitter

This emitter gets called when an empty block gets send by Geany. It generates a template to document the source file and sends it to the output stream.

Definition at line 224 of file fb-doc_emit_doxy.bas.

Here is the caller graph for this function:



15.17.3 Variable Documentation

15.17.3.1 const VAR DOXY_START = NL & "/'∗ "

the start of a comment block

Definition at line 32 of file fb-doc_emit_doxy.bas.

15.17.3.2 const VAR DOXY_END

Initial value:

```
= NL &
                NL & FIXME &
                     COMM_END
```

the end of a comment block

Definition at line 33 of file fb-doc_emit_doxy.bas.

15.18 fb-doc_emit_doxy.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emit_doxy.bas
00002 \brief Emitter for Doxygen templates
00003
00004 This file contains emitter functions for the \ref EmitterIF to
00005 generate template blocks for the Doxygen Back-end. It's designed to
00006 be used in Geany mode (see section \ref subsectExaDoxy for an example).
00007
00008 The emitters return all original source code unchanged. Additionally
00009 relevant lines (or code blocks) get prepended by a multi line block
00010 of documentation. This works for `SUB`s, `FUNCTION`s, `TYPE, UNION` and
00011 `ENUM` blocks, declarations (`VAR  DIM  CONST  COMMON  EXTERN
00012 STATIC`) and \#`DEFINE`s or \#`MACRO`s.
00013
00014 The block starts with the C declaration of the construct and a line
00015 for the brief description. Then the members are listed with leading
00016 param or var keyword and followed by a line for the brief
00017 description. The block ends by a `FIXME` text for the detailed
00018 description.
00019
00020 Since the prefered way to document variables with Doxygen is to
00021 write the comment behind the variable, this emitter is mostly
00022 helpful for documenting functions and their parameter lists.
00023
00024 When an empty line is send by Geany, the text of the \ref doxy_empty()
00025 function gets emitted. This is a comment block template to describe
00026 the source file.
00027
00028 '/
00029
00030
00031 CONST _
00032    DOXY_START = NL & "/'* ", _ '*< the start of a comment block
00033      DOXY_END = NL & _
00034                 NL & FIXME & _
00035                      COMM_END  '*< the end of a comment block
00036
00037
00038 /'* \brief Emitter to generate a line for a parameter list entry
00039 \param P the parser calling this emitter
00040
00041 This emitter gets called when the parser is in a parameter list of a
00042 function (`SUB  FUNCTION  PROPERTY  CONSTRUCTOR  DESTRUCTOR`). It
00043 generates a line for each parameter and sends it (them) to the
00044 output stream.
00045
00046 '/
00047 SUB doxy_entryListPara CDECL(BYVAL P AS Parser PTR)
00048   WITH *P
00049     IF .NamTok THEN Code(NL & "\param " & .SubStr(.NamTok) & " " & FIXME)
00050   END WITH
00051 END SUB
00052
00053
00054 /'* \brief Emitter to generate a template for a function
00055 \param P the parser calling this emitter
00056
00057 This emitter gets called when the parser finds a function (`SUB
00058 FUNCTION  PROPERTY  CONSTRUCTOR  DESTRUCTOR`). It generates a
00059 template to document the function and its parameter list and
00060 sends it to the output stream.
00061
00062 '/
00063 SUB doxy_func_ CDECL(BYVAL P AS Parser PTR)
00064   WITH *P
00065     VAR a = .StaTok[1], b = .ParTok, t = .TypTok
00066     cEmitSource(P, .StaTok[1])
00067     Code(     DOXY_START & "\fn ")
00068     OPT->CreateFunction(P)
00069     Code(NL & "\brief " & FIXME)
00070     IF b THEN .ParTok = b : .parseListPara(@doxy_entryListPara)
00071 '& doxy_entryListPara(); // pseudo function call (helps Doxygen documenting the interface)
00072     IF t THEN Code(NL & "\returns " & FIXME)
00073     Code(DOXY_END)
00074     .SrcBgn = a
00075   END WITH
00076 END SUB
```

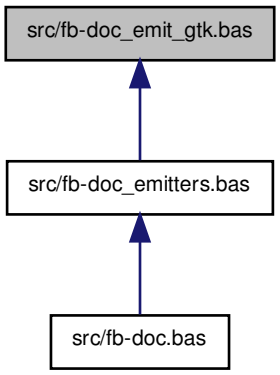
```
00077
00078
00079 /'* \brief Emitter to generate a template for a declaration
00080 \param P the parser calling this emitter
00081
00082 This emitter gets called when the parser is in a declaration (`VAR
00083 DIM  CONST  COMMON  EXTERN  STATIC`). It generates a line for
00084 each variable name and sends it (them) to the output stream.
00085
00086 '/
00087 SUB doxy_decl_ CDECL(BYVAL P AS Parser PTR)
00088   WITH *P
00089     IF 0 = .ListCount THEN
00090       cEmitSource(P, .StaTok[1])
00091       Code(DOXY_START)
00092
00093       IF .FunTok THEN  Code("\fn ") _
00094                  ELSE  Code("\var ")
00095
00096       SELECT CASE AS CONST *.StaTok
00097       CASE .TOK_CONS,.TOK_STAT, .TOK_COMM, .TOK_EXRN
00098         IF OPT->Types = OPT->C_STYLE THEN Code(LCASE(.SubStr(.StaTok)) & " ") _
00099                                      ELSE Code(      .SubStr(.StaTok)  & "_")
00100       CASE .TOK_TYPE : Code("typedef ")
00101         IF 0 = .FunTok ANDALSO .TypTok > .NamTok THEN Code("struct ")
00102       END SELECT
00103     END IF
00104
00105     IF     .FunTok THEN
00106       VAR a = .SrcBgn, b = .ParTok, t = .TypTok
00107       OPT->CreateFunction(P)
00108       Code(NL & "\brief " & FIXME)
00109       IF b THEN .ParTok = b : .parseListPara(@doxy_entryListPara)
00110 '& doxy_entryListPara(); // pseudo function call (helps Doxygen documenting the interface)
00111       IF t THEN Code(NL & "\returns: " & FIXME)
00112       Code(DOXY_END)
00113       .SrcBgn = a
00114       Code("'' " & PROG_NAME & "-hint: consider to document the functions body instead." & NL)
00115     ELSEIF .TypTok THEN
00116       OPT->CreateVariable(P)
00117       Code(NL & "\brief " & FIXME & DOXY_END)
00118     ELSE
00119       IF 0 = .ListCount THEN Code("VAR ")
00120                              Code(.SubStr(.NamTok))
00121       'IF .IniTok THEN        Code(" " & .VarIni)
00122       IF .IniTok THEN        cIni(P)
00123       Code(NL & "\brief " & FIXME & DOXY_END)
00124     END IF
00125   END WITH
00126 END SUB
00127
00128
00129 /'* \brief Emitter to generate a templates for a macro
00130 \param P the parser calling this emitter
00131
00132 This emitter gets called when the parser finds a macro (\#`DEFINE`
00133 \#`MACRO`). It generates a template to document the macro and sends it
00134 to the output stream.
00135
00136 '/
00137 SUB doxy_defi_ CDECL(BYVAL P AS Parser PTR)
00138   WITH *P
00139     cEmitSource(P, .StaTok[1])
00140     Code(  DOXY_START & "\def " & .SubStr(.NamTok) & _
00141       NL & "\brief " & FIXME & _
00142            DOXY_END)
00143   END WITH
00144 END SUB
00145
00146
00147 /'* \brief Emitter to generate a line for a block entry
00148 \param P the parser calling this emitter
00149
00150 This emitter gets called when the parser is in a block (`TYPE  ENUM
00151 UNION`). It generates a line for each member and sends it (them) to
00152 the output stream.
00153
00154 '/
00155 SUB doxy_emitBlockNames CDECL(BYVAL P AS Parser PTR)
00156   WITH *P
00157     SELECT CASE AS CONST *.Tk1
00158     CASE .TOK_PRIV, .TOK_PROT ': .SrcBgn = 0 ' !!! ToDo: hide private?
00159     CASE .TOK_PUBL            ': .SrcBgn = 1
00160     CASE .TOK_CLAS, .TOK_TYPE, .TOK_UNIO
00161       .parseBlockTyUn(@doxy_emitBlockNames)
00162     CASE .TOK_ENUM
00163       .parseBlockEnum(@doxy_emitBlockNames)
```

```
00164     CASE ELSE : IF 0 = .NamTok THEN EXIT SUB
00165       Code(NL & "\var " & .BlockNam & "::" & .SubStr(.NamTok) &  " " & FIXME & _
00166            NL & "\brief " & FIXME)
00167     END SELECT
00168 '& doxy_emitBlockNames(); // pseudo function call (helps Doxygen documenting the interface)
00169   END WITH
00170 END SUB
00171
00172
00173 /'* \brief Emitter to generate templates for blocks
00174 \param P the parser calling this emitter
00175
00176 This emitter gets called when the parser finds a block (`TYPE  UNION
00177 ENUM`). It generates a template to document the block with one line
00178 for each member and sends it to the output stream.
00179
00180 '/
00181 SUB doxy_Block CDECL(BYVAL P AS Parser PTR)
00182   WITH *P
00183     cEmitSource(P, .StaTok[1])
00184     SELECT CASE AS CONST *.Tk1
00185     CASE .TOK_ENUM
00186       Code(  DOXY_START & "\enum " & .BlockNam & _
00187         NL & "\brief " & FIXME & _
00188         NL & _
00189         NL & FIXME & _
00190         NL)
00191       .parseBlockEnum(@doxy_emitBlockNames)
00192     CASE .TOK_UNIO
00193       Code(  DOXY_START & "\union " & .BlockNam & _
00194         NL & "\brief " & FIXME & _
00195         NL & _
00196         NL & FIXME & _
00197         NL)
00198       .parseBlockTyUn(@doxy_emitBlockNames)
00199     CASE ELSE
00200       IF OPT->Types = OPT->FB_STYLE _
00201         THEN Code(DOXY_START & "\class ") _
00202         ELSE Code(DOXY_START & "\struct ")
00203       Code(  .BlockNam & _
00204         NL & "\brief " & FIXME & _
00205         NL & _
00206         NL & FIXME & _
00207         NL)
00208       .parseBlockTyUn(@doxy_emitBlockNames)
00209     END SELECT
00210 '& doxy_emitBlockNames(); // pseudo function call (helps Doxygen documenting the interface)
00211     Code(COMM_END)
00212   END WITH
00213 END SUB
00214
00215
00216 /'* \brief Emitter for an empty Geany block
00217 \param P the parser calling this emitter
00218
00219 This emitter gets called when an empty block gets send by Geany. It
00220 generates a template to document the source file and sends it to the
00221 output stream.
00222
00223 '/
00224 SUB doxy_empty CDECL(BYVAL P AS Parser PTR)
00225   WITH *P
00226     Code(  DOXY_START & "\file " & FIXME & _
00227       NL & "\brief " & FIXME & _
00228            DOXY_END)
00229   END WITH
00230 END SUB
00231
00232
00233 WITH_NEW_EMITTER("DoxygenTemplates")
00234    .Error_ = @c_error '           we use the standard error emitter here
00235
00236     .Func_ = @doxy_func_
00237     .Decl_ = @doxy_decl_
00238     .Defi_ = @doxy_defi_
00239     .Enum_ = @doxy_Block
00240     .Unio_ = @doxy_Block
00241     .Clas_ = @doxy_Block
00242     .Init_ = @geanyInit '        ... and the Geany init / exit functions
00243     .Exit_ = @geanyExit
00244    .Empty_ = @doxy_empty
00245 END WITH
00246
00247
```

15.19 src/fb-doc_emit_gtk.bas File Reference

Emitter for gtk-doc templates.

This graph shows which files directly or indirectly include this file:



Functions

- SUB_CDECL gtk_emit_Name (BYVAL_AS_Parser_PTR P)

  Emitter to generate a name line.
- SUB_CDECL gtk_defi_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a macro template.
- SUB_CDECL gtk_decl_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a template for a declaration.
- SUB_CDECL gtk_func_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a template for a function.
- SUB_CDECL gtk_emitBlockNames (BYVAL_AS_Parser_PTR P)

  Emitter to generate a line for a block entry.
- SUB_CDECL gtk_Block (BYVAL_AS_Parser_PTR P)

  Emitter to generate templates for blocks.
- SUB_CDECL gtk_empty (BYVAL_AS_Parser_PTR P)

  Emitter for an empty Geany block.

Variables

- const VAR SINCE = NL & NL & "Since: 0.0"

  text added at each block end
- const VAR GTK_START = "/* "

  the start of a comment block
- const VAR GTK_END

  the end of a comment block

15.19.1 Detailed Description

Emitter for gtk-doc templates. This file contains the emitter for gtk-doc templates. The emitter is designed to create documentation comment blocks for the gtk-doc back-and and it's designed to be used in Geany mode (see section Gtk-doc for an example).

The emitter returns all original source code. Relevant lines (or code blocks) get prepended by a multi line block of documentation. This works for SUBs/FUNCTIONs, TYPE, UNION and ENUM blocks and #DEFINEs / #MACROs.

The first line of the comment block contains the name of the construct, appended by a colon. Then the members are listed with a leading @ character (parameters in case of a SUB FUNCTION or member variables in case of a block). The block ends by a FIXME text and a line with a since keyword.

When an empty line is send by Geany, a comment block template to describe the file gets emitted.

Definition in file fb-doc_emit_gtk.bas.

15.19.2 Function Documentation

15.19.2.1 SUB_CDECL gtk_emit_Name (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a name line.

Parameters

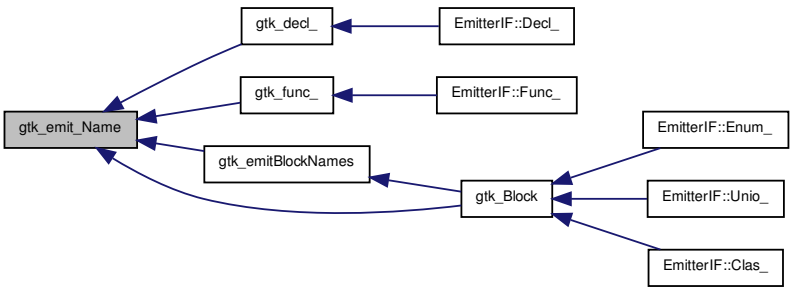
P	the parser calling this emitter

Generate a name for a gtk-doc template. Used in lists (parameters or variable declarations) or blocks (ENUM, TYPE, UNION). It generates a line to document the variable and sends it to the output stream.

Definition at line 43 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:

Here is the caller graph for this function:



15.19.2.2 SUB_CDECL gtk_defi_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a macro template.

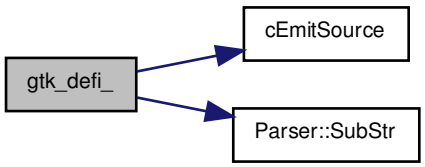
Parameters

P	the parser calling this emitter

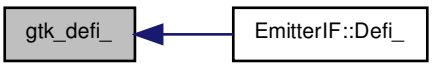
This emitter gets called when the parser finds a macro (#DEFINE / #MACRO). It generates a template to document the macro and sends it to the output stream.

Definition at line 58 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.3 SUB_CDECL gtk_decl_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a template for a declaration.

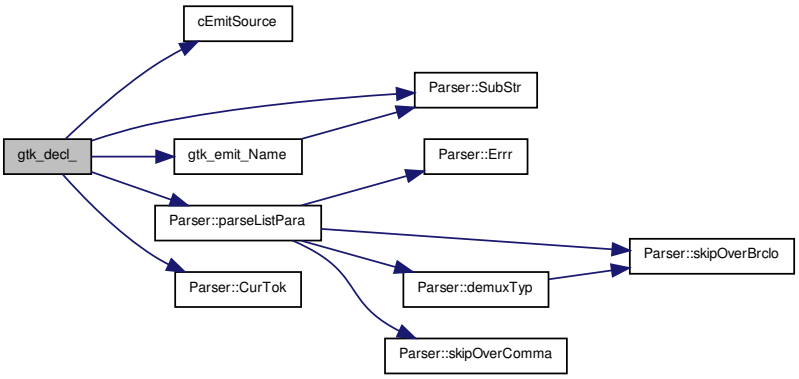
Parameters

P	the parser calling this emitter

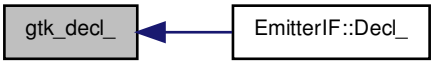
This emitter gets called when the parser is in a declaration (VAR DIM CONST COMMON EXTERN STATIC). It generates a line for each variable name and sends it (them) to the output stream.

Definition at line 74 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.4 SUB_CDECL gtk_func_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a template for a function.

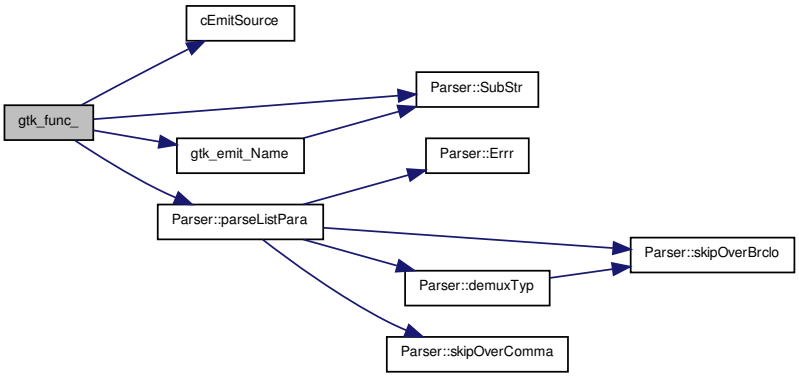
Parameters

P	the parser calling this emitter

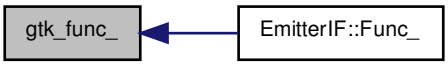
This emitter gets called when the parser finds a function (`SUB FUNCTION PROPERTY CONSTRUCTOR DESTRUCTOR`). It generates a template to document the function and its parameter list and sends it to the output stream.

Definition at line 103 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.5 SUB_CDECL gtk_emitBlockNames (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a line for a block entry.

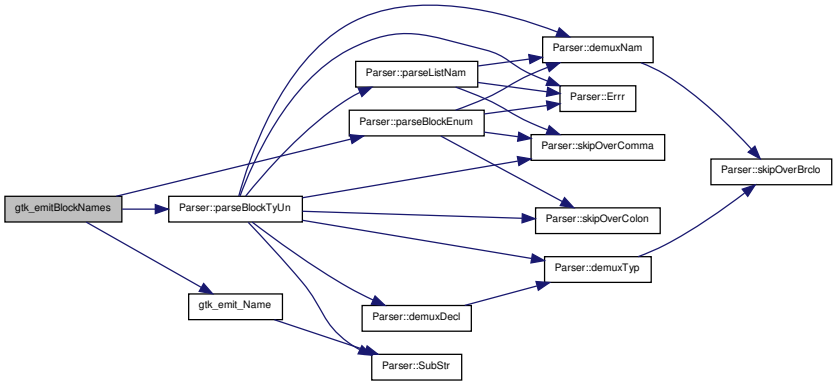
Parameters

P	the parser calling this emitter

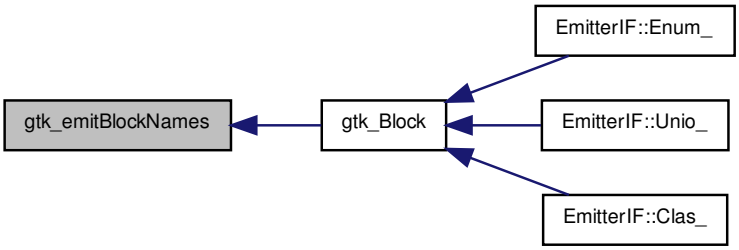
This emitter gets called when the parser is in a block (`TYPE ENUM UNION`). It generates a line for each member and sends it (them) to the output stream.

Definition at line 131 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.6 SUB_CDECL gtk_Block (BYVAL_AS_Parser_PTR *P*)

Emitter to generate templates for blocks.

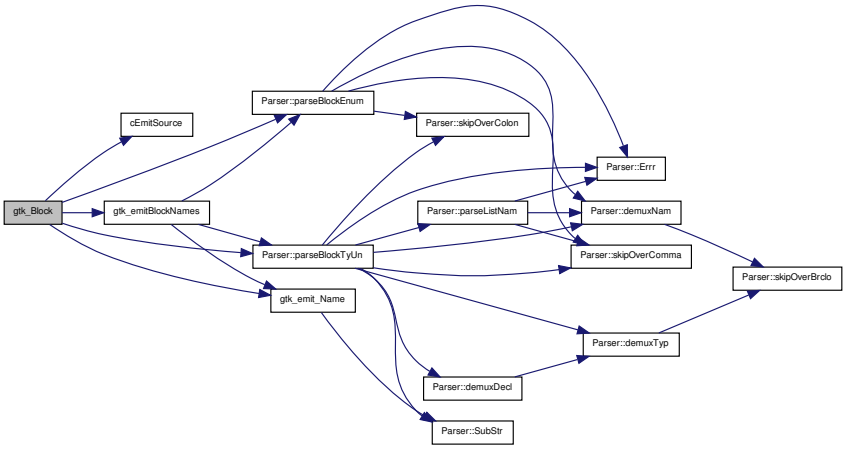
Parameters

P	the parser calling this emitter

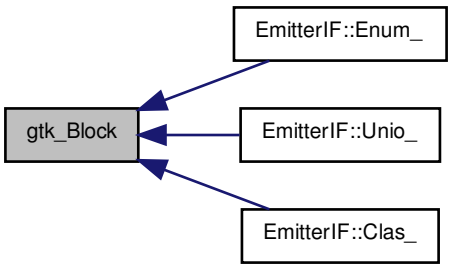
This emitter gets called when the parser finds a block (`TYPE UNION ENUM`). It generates a template to document the block with one line for each member and sends it to the output stream.

Definition at line 156 of file fb-doc_emit_gtk.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.7 SUB_CDECL gtk_empty (BYVAL_AS_Parser_PTR *P*)

Emitter for an empty Geany block.

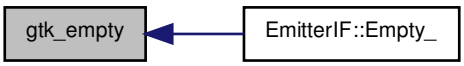
Parameters

P	the parser calling this emitter

This emitter gets called when an empty block gets send by Geany. It generates a template to document the source file and sends it to the output stream.

Definition at line 182 of file fb-doc_emit_gtk.bas.

Here is the caller graph for this function:



15.19.3 Variable Documentation

15.19.3.1 const VAR SINCE = NL & NL & "Since: 0.0"

text added at each block end

Definition at line 27 of file fb-doc_emit_gtk.bas.

15.19.3.2 const VAR GTK_START = "/'* "

the start of a comment block

Definition at line 28 of file fb-doc_emit_gtk.bas.

15.19.3.3 const VAR GTK_END

Initial value:

```
= NL &
                NL & FIXME &
                     SINCE &
                     COMM_END
```

the end of a comment block

Definition at line 29 of file fb-doc_emit_gtk.bas.

15.20 fb-doc_emit_gtk.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emit_gtk.bas
00002 \brief Emitter for gtk-doc templates
00003
00004 This file contains the emitter for gtk-doc templates. The emitter is
00005 designed to create documentation comment blocks for the gtk-doc
00006 back-and and it's designed to be used in Geany mode (see section \ref
00007 subsectExaGtkdoc for an example).
00008
00009 The emitter returns all original source code. Relevant lines (or
00010 code blocks) get prepended by a multi line block of documentation.
00011 This works for `SUB`s/`FUNCTION`s, `TYPE, UNION` and `ENUM` blocks and
00012 \#`DEFINE`s / \#`MACRO`s.
00013
00014 The first line of the comment block contains the name of the
00015 construct, appended by a colon. Then the members are listed with a
00016 leading @ character (parameters in case of a `SUB  FUNCTION` or
00017 member variables in case of a block). The block ends by a `FIXME`
00018 text and a line with a since keyword.
00019
00020 When an empty line is send by Geany, a comment block template to
00021 describe the file gets emitted.
00022
```

```
00023 ''/
00024
00025
00026 CONST _
00027       SINCE = NL & NL & "Since: 0.0", _ '*< text added at each block end
00028   GTK_START =           "/'* ", _       '*< the start of a comment block
00029     GTK_END = NL & _
00030               NL & FIXME & _
00031                    SINCE & _
00032                    COMM_END             '*< the end of a comment block
00033
00034
00035 /'* \brief Emitter to generate a name line
00036 \param P the parser calling this emitter
00037
00038 Generate a name for a gtk-doc template. Used in lists (parameters or
00039 variable declarations) or blocks (`ENUM, TYPE, UNION`). It generates a
00040 line to document the variable and sends it to the output stream.
00041
00042 ''/
00043 SUB gtk_emit_Name CDECL(BYVAL P AS Parser PTR)
00044   WITH *P
00045     IF .NamTok THEN Code(NL & "@" & P->SubStr(P->NamTok) & ": " & FIXME)
00046   END WITH
00047 END SUB
00048
00049
00050 /'* \brief Emitter to generate a macro template
00051 \param P the parser calling this emitter
00052
00053 This emitter gets called when the parser finds a macro (\#`DEFINE` /
00054 \#`MACRO`). It generates a template to document the macro and sends it
00055 to the output stream.
00056
00057 ''/
00058 SUB gtk_defi_ CDECL(BYVAL P AS Parser PTR)
00059   WITH *P
00060     cEmitSource(P, .StaTok[1])
00061     Code(GTK_START & .SubStr(.NamTok) & ":" & GTK_END)
00062   END WITH
00063 END SUB
00064
00065
00066 /'* \brief Emitter to generate a template for a declaration
00067 \param P the parser calling this emitter
00068
00069 This emitter gets called when the parser is in a declaration (`VAR
00070 DIM  CONST  COMMON  EXTERN  STATIC`). It generates a line for
00071 each variable name and sends it (them) to the output stream.
00072
00073 ''/
00074 SUB gtk_decl_ CDECL(BYVAL P AS Parser PTR)
00075   WITH *P
00076     IF 0 = .ListCount THEN
00077       cEmitSource(P, .Tk1[1])
00078       Code(GTK_START & .SubStr(.NamTok) & ":")
00079     END IF
00080
00081     IF 0 = .FunTok THEN gtk_emit_Name(P) _
00082                    ELSE IF .ParTok THEN .parseListPara(@gtk_emit_Name)
00083 '& gtk_emit_Name(); // pseudo function call (helps Doxygen documenting the interface)
00084
00085     IF *.CurTok > .TOK_EOS THEN EXIT SUB
00086     Code(GTK_END)
00087
00088     IF 0 = .FunTok THEN EXIT SUB
00089     Code("'' " & PROG_NAME & "-hint: consider to document the functions body instead." & NL)
00090   END WITH
00091 END SUB
00092
00093
00094 /'* \brief Emitter to generate a template for a function
00095 \param P the parser calling this emitter
00096
00097 This emitter gets called when the parser finds a function (`SUB
00098 FUNCTION  PROPERTY  CONSTRUCTOR  DESTRUCTOR`). It generates a
00099 template to document the function and its parameter list and
00100 sends it to the output stream.
00101
00102 ''/
00103 SUB gtk_func_ CDECL(BYVAL P AS Parser PTR) ' !!! ToDo member functions
00104   WITH *P
00105     VAR t = .TypTok
00106     cEmitSource(P, .StaTok[1])
00107     Code(GTK_START & .SubStr(.NamTok) & ":")
00108     IF .ParTok THEN .parseListPara(@gtk_emit_Name)
00109 '& gtk_emit_Name(); // pseudo function call (helps Doxygen documenting the interface)
```

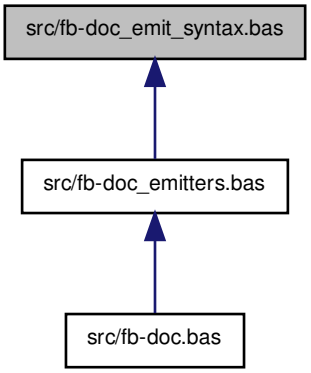
```
00110
00111     Code( _
00112         NL & _
00113         NL & FIXME)
00114     IF t THEN Code( _
00115         NL & _
00116         NL & "Returns: " & FIXME)
00117     Code(     SINCE & _
00118               COMM_END)
00119   END WITH
00120 END SUB
00121
00122
00123 /'* \brief Emitter to generate a line for a block entry
00124 \param P the parser calling this emitter
00125
00126 This emitter gets called when the parser is in a block (`TYPE  ENUM
00127 UNION`). It generates a line for each member and sends it (them) to
00128 the output stream.
00129
00130 '/
00131 SUB gtk_emitBlockNames CDECL(BYVAL P AS Parser PTR)
00132   WITH *P
00133     SELECT CASE AS CONST *.Tk1
00134     CASE .TOK_PRIV, .TOK_PROT ': .SrcBgn = 0 ' !!! ToDo: hide private?
00135     CASE .TOK_PUBL            ': .SrcBgn = 1
00136     CASE .TOK_CLAS, .TOK_TYPE, .TOK_UNIO
00137       .parseBlockTyUn(@gtk_emitBlockNames)
00138 '& gtk_emitBlockNames(); // pseudo function call (helps Doxygen documenting the interface)
00139     CASE .TOK_ENUM
00140       .parseBlockEnum(@gtk_emit_Name)
00141     CASE ELSE : IF 0 = .NamTok THEN EXIT SUB
00142       gtk_emit_Name(P)
00143     END SELECT
00144   END WITH
00145 END SUB
00146
00147
00148 /'* \brief Emitter to generate templates for blocks
00149 \param P the parser calling this emitter
00150
00151 This emitter gets called when the parser finds a block (`TYPE  UNION
00152 ENUM`). It generates a template to document the block with one line
00153 for each member and sends it to the output stream.
00154
00155 '/
00156 SUB gtk_Block CDECL(BYVAL P AS Parser PTR)
00157   WITH *P
00158     cEmitSource(P, .StaTok[1])
00159     Code( GTK_START)
00160     IF LEN(.BlockNam) THEN Code(.BlockNam & ":")
00161
00162     SELECT CASE AS CONST *.Tk1
00163     CASE .TOK_ENUM : .parseBlockEnum(@gtk_emit_Name)
00164     CASE ELSE :      .parseBlockTyUn(@gtk_emitBlockNames)
00165     END SELECT
00166 '& gtk_emit_Name(); // pseudo function calls (help Doxygen documenting the interface)
00167 '& gtk_emitBlockNames();
00168
00169     Code(GTK_END)
00170   END WITH
00171 END SUB
00172
00173
00174 /'* \brief Emitter for an empty Geany block
00175 \param P the parser calling this emitter
00176
00177 This emitter gets called when an empty block gets send by Geany. It
00178 generates a template to document the source file and sends it to the
00179 output stream.
00180
00181 '/
00182 SUB gtk_empty CDECL(BYVAL P AS Parser PTR)
00183   WITH *P
00184     Code(  GTK_START & _
00185            "SECTION: " & FIXME & _
00186       NL & "@short_description: " & FIXME & _
00187       NL & "@title: " & FIXME & _
00188       NL & "@section_id: " & FIXME & _
00189       NL & "@see_also: " & FIXME & _
00190       NL & "@stability: " & FIXME & _
00191       NL & "@include: " & FIXME & _
00192       NL & "@image: " & FIXME & _
00193            GTK_END & _
00194       NL)
00195   END WITH
00196 END SUB
```

```
00197
00198
00199 WITH_NEW_EMITTER("GtkDocTemplates")
00200    .Error_ = @c_error  '*< we use the standard error emitter here
00201
00202     .Func_ = @gtk_func_
00203     .Decl_ = @gtk_decl_
00204     .Defi_ = @gtk_defi_
00205     .Enum_ = @gtk_Block
00206     .Unio_ = @gtk_Block
00207     .Clas_ = @gtk_Block
00208     .Init_ = @geanyInit
00209     .Exit_ = @geanyExit
00210    .Empty_ = @gtk_empty
00211 END WITH
00212
00213
```

15.21 src/fb-doc_emit_syntax.bas File Reference

Emitter for repairing the Doxygen syntax highlighting.

This graph shows which files directly or indirectly include this file:



Classes

- class RepData

  A container for string replacements.
- class Highlighter

  Class to process the syntax highlighting.
- union Highlighter.__unnamed__
- class Highlighter.__unnamed__.__unnamed__

Macros

- #define GET_WORD_TYPE()

  check a word in source code

Functions

- FUNCTION_AS_STRING html_specials (BYVAL_AS_UBYTE_PTR T, BYVAL_AS_INTEGER A, BYVAL_AS_INTEGER L)

  Replace special characters for HTML output.
- FUNCTION_AS_STRING tex_specials (BYVAL_AS_UBYTE_PTR T, BYVAL_AS_INTEGER A, BYVAL_AS_INTEGER L)

  Replace special characters for LaTeX output.
- FUNCTION_AS_STRING xml_specials (BYVAL_AS_UBYTE_PTR T, BYVAL_AS_INTEGER A, BYVAL_AS_INTEGER L)

  Replace special characters for XML output.
- FUNCTION_AS_STRING html_eol (BYVAL_AS_RepData_PTR Symb, BYREF_AS_INTEGER Nr)

  Generate end of line code for HTML output.
- FUNCTION_AS_STRING tex_eol (BYVAL_AS_RepData_PTR Symb, BYREF_AS_INTEGER Nr)

  Generate end of line code for LaTeX output.
- FUNCTION_AS_STRING xml_eol (BYVAL_AS_RepData_PTR Symb, BYREF_AS_INTEGER Nr)

  Generate end of line code for XML output.
- SUB_CDECL synt_init (BYVAL_AS_Parser_PTR P)

  Emitter to be called per file before parsing starts.
- SUB_CDECL synt_exit (BYVAL_AS_Parser_PTR P)

  Emitter to be called after parsing of a file.
- SUB_CDECL synt_incl (BYVAL_AS_Parser_PTR P)

  Emitter to generate an include statement.
- SUB_CDECL synt_func_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a function name.

15.21.1 Detailed Description

Emitter for repairing the Doxygen syntax highlighting. This file contains the emitter for html syntax highlighting. The emitter is designed to create code listings with high-linghting tags defined by Doxygen. This emitter reads the original Doxygen output files and replaces the code section. This works for html, LaTeX and XML output.

The code includes links to the documentation. These links are extracted from the original Doxygen output. Since the names in the link tags change the repairing process can be done only once (ie the name of the constructor *Parser::Parser* in intermediate format gets just *Parser* in FB source).

Definition in file fb-doc_emit_syntax.bas.

15.21.2 Class Documentation

15.21.2.1 union Highlighter.__unnamed__

Definition at line 291 of file fb-doc_emit_syntax.bas.

Class Members

__unnamed__	__unnamed__	
INTEGER	GenAny	All output flags.

15.21.2.2 class Highlighter.__unnamed__.__unnamed__

Definition at line 292 of file fb-doc_emit_syntax.bas.

Class Members

INTEGER	GenHtml: 1	Flag for html output.
INTEGER	GenTex: 1	Flag for LaTeX output.
INTEGER	GenXml: 1	Flag for XML output.

15.21.3 Macro Definition Documentation

15.21.3.1 #define GET_WORD_TYPE( )

Value:

```
/* (multi line FreeBasic #MACRO)
  VAR word = MID(*T, start + 1, size)
  VAR res = IIF(Symbols, Symbols->rep(word), SADD(word))
  typ = FB_CODE
  IF res = SADD(word) THEN '     no symbol, check keyword & preprocessor
    word = UCASE(word)
    res = word_type(word)
    typ = IIF(res, ASC(*res), FB_CODE)

    SELECT CASE AS CONST OPT->CaseMode '  reformat letter cases, if required
    CASE OPT->CASE_LOWER : MID(*T, start + 1, size) = LCASE(word)
    CASE OPT->CASE_MIXED : MID(*T, start + 1, size) = MID(*res, 2)
    CASE OPT->CASE_UPPER : MID(*T, start + 1, size) = word
    END SELECT
  ELSE
    typ = CAST(INTEGER, res) '               start of replacement string
  END IF
#ENDMACRO */
```

check a word in source code

This macro is used for single source reasons. It reads the current word from the input buffer and checks if it matches

1. an entry in the container Highlighter::Symbols (links red from orig. file),
2. the FB keyword list

The result is in the typ variable:

1. = keyword,
2. = keywordtype,
3. = preprocessor,
4. or greater = link (type gets a ZSTRING PTR to the context)

Definition at line 1178 of file fb-doc_emit_syntax.bas.

15.21.4 Function Documentation

15.21.4.1 FUNCTION_AS_STRING html_specials (BYVAL_AS_UBYTE_PTR *T*, BYVAL_AS_INTEGER *A*, BYVAL_AS_INTEGER *L*)

Replace special characters for HTML output.

Parameters

T	The Buffer to read from
A	Start position (zero based)
L	Length of substring

Returns

A string with replaced special characters

The function is used as Highlighter::special_chars() function. It extracts a substring from the input buffer. Special characters are replaced by their HTML equivalents.

Definition at line 88 of file fb-doc_emit_syntax.bas.

15.21.4.2 FUNCTION_AS_STRING tex_specials (BYVAL_AS_UBYTE_PTR *T*, BYVAL_AS_INTEGER *A*, BYVAL_AS_INTEGER *L*)

Replace special characters for LaTeX output.

Parameters

T	The Buffer to read from
A	Start position (zero based)
L	Length of substring

Returns

A string with replaced special characters

The function is used as Highlighter::special_chars() function. It extracts a substring from the input buffer. Special characters are replaced by their LaTeX equivalents.

Definition at line 114 of file fb-doc_emit_syntax.bas.

15.21.4.3 FUNCTION_AS_STRING xml_specials (BYVAL_AS_UBYTE_PTR *T*, BYVAL_AS_INTEGER *A*, BYVAL_AS_INTEGER *L*)

Replace special characters for XML output.

Parameters

T	The Buffer to read from
A	Start position (zero based)
L	Length of substring

Returns

A string with replaced special characters

The function is used as Highlighter::special_chars() function. It extracts a substring from the input buffer. Special characters are replaced by their XML equivalents.

Definition at line 142 of file fb-doc_emit_syntax.bas.

15.21.4.4 FUNCTION_AS_STRING html_eol (BYVAL_AS_RepData_PTR *Symb*, BYREF_AS_INTEGER *Nr*)

Generate end of line code for HTML output.

Parameters

Symb	Symbol table for cross-referencing
Nr	Line number

Returns

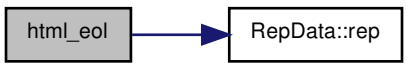
A string to end the current line and start a new one

The function is used as Highlighter::eol() function. It generates code to end the current line and start a new one with the next line number. The number counter gets increased and returned as a BYREF parameter. Special line no. are

- 1: the first line (emit no line end but the line start)
- 0: the last line (emit the line end but no line start)

Definition at line 174 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



15.21.4.5 FUNCTION_AS_STRING tex_eol (BYVAL_AS_RepData_PTR *Symb*, BYREF_AS_INTEGER *Nr*)

Generate end of line code for LaTeX output.

Parameters

Symb	Symbol table for cross-referencing
Nr	Line number

Returns

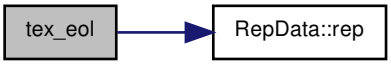
A string to end the current line and start a new one

The function is used as Highlighter::eol() function. It generates code to end the current line and start a new one with the next line number. The number counter gets increased and returned as a BYREF parameter. Special line no. are

- 1: the first line (emit no line end but the line start)
- 0: the last line (emit the line end but no line start)

Definition at line 204 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



15.21.4.6 FUNCTION_AS_STRING xml_eol (BYVAL_AS_RepData_PTR *Symb*, BYREF_AS_INTEGER *Nr*)

Generate end of line code for XML output.

Parameters

Symb	Symbol table for cross-referencing
Nr	Line number

Returns

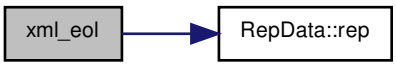
A string to end the current line and start a new one

The function is used as Highlighter::eol() function. It generates code to end the current line and start a new one with the next line number. The number counter gets increased and returned as a BYREF parameter. Special line numbers are

- 1: the first line (emit no line end but the line start)
- 0: the last line (emit the line end but no line start)

Definition at line 230 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



15.21.4.7 SUB_CDECL synt_init (BYVAL_AS_Parser_PTR *P*)

Emitter to be called per file before parsing starts.

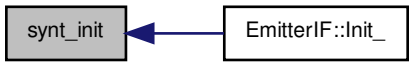
Parameters

P	the parser calling this emitter

This emitter gets called before the parser starts its parsing process. It initializes the FB source code emission.

Definition at line 1452 of file fb-doc_emit_syntax.bas.

Here is the caller graph for this function:



15.21.4.8 SUB_CDECL synt_exit (BYVAL_AS_Parser_PTR *P*)

Emitter to be called after parsing of a file.

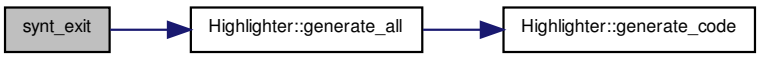
Parameters

P	the parser calling this emitter

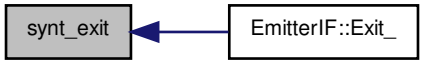
This emitter gets called after the parser ends its parsing process. It sends the rest of the FB source code to the output stream.

Definition at line 1475 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.21.4.9 SUB_CDECL synt_incl (BYVAL_AS_Parser_PTR *P*)

Emitter to generate an include statement.

Parameters

P	the parser calling this emitter

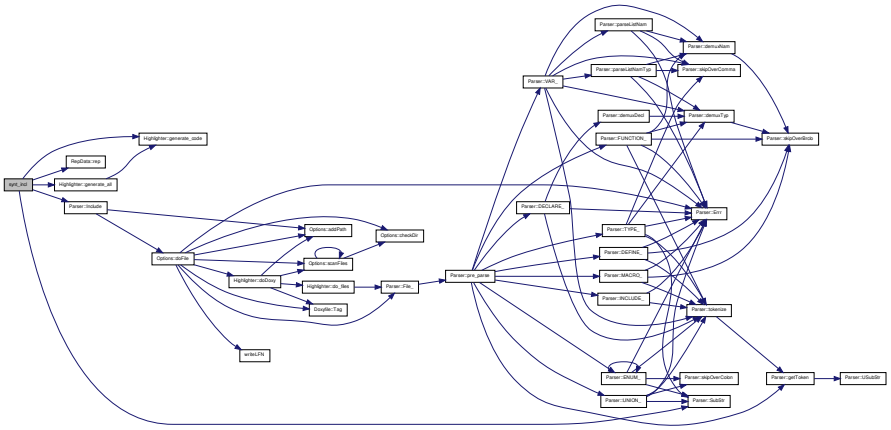
This emitter operates on include statements. It extracts the file name and checks the Highlighter::Symbols table for a matching link. If there is no link, nothing is done.

In case of a matching link the source code gets emitted up to the link.

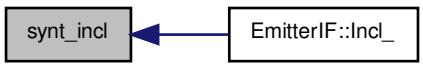
File names need special handling because the string literals don't get checked for linkage.

Definition at line 1500 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.21.4.10 SUB_CDECL synt_func_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a function name.

Parameters

P	the parser calling this emitter

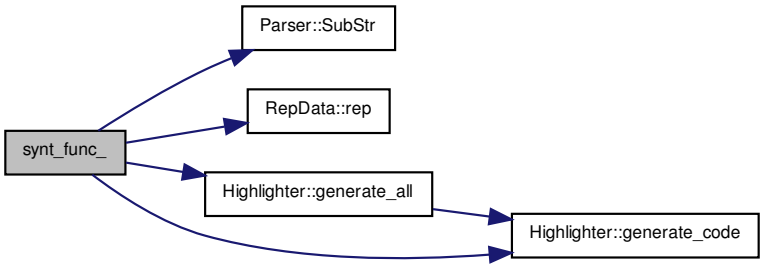
This emitter operates on SUB / FUNCTION / PROPERTY definitions (function body, not declaration). It extracts the function name and checks the Highlighter::Symbols table for a matching link. If there is no link, nothing is done.

In case of a matching link the source code gets emitted up to the link.

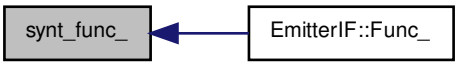
Funktion names need special handling because the names of CONSTRUCTORs and DESTRUCTORs vary between intermediate format and FB source.

Definition at line 1534 of file fb-doc_emit_syntax.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.22 fb-doc_emit_syntax.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emit_syntax.bas
00002 \brief Emitter for repairing the Doxygen syntax highlighting.
00003
00004 This file contains the emitter for html syntax highlighting. The
00005 emitter is designed to create code listings with high-linghting tags
00006 defined by Doxygen. This emitter reads the original Doxygen output
00007 files and replaces the code section. This works for html, LaTeX and XML
00008 output.
00009
00010 The code includes links to the documentation. These links are extracted
00011 from the original Doxygen output. Since the names in the link tags
00012 change the repairing process can be done only once (ie the name of the
00013 constructor *Parser::Parser* in intermediate format gets just *Parser*
00014 in FB source).
00015
00016 '/
00017
00018
00019 /'* \brief A container for string replacements
00020
00021 This class is to store two tables, one of search strings and a second
00022 one of their replacements. It's used to collect the links from the
00023 original source and replace symbol names in the emitter output, as well
00024 as referenced line numbers and \#`INCLUDE` files.
00025
00026 '/
00027 TYPE RepData
00028   AS STRING _
00029       O _                  '*< The output (a list of counters and strings)
00030     , I = MKI(0) & CHR(1) '*< The input (a list of strings to search for)
00031   DECLARE FUNCTION add(BYREF AS STRING, BYREF AS STRING) AS ZSTRING PTR
00032   DECLARE FUNCTION rep(BYREF AS STRING) AS ZSTRING PTR
00033 END TYPE
00034
00035 /'* \brief Add a new element pair
00036 \param S The string to search for
```

```
00037 \param R The string to replace with
00038 \returns zero on success, otherwise a ZSTRING PTR to an error code
00039
00040 Add a new pair of strings to the container tables. The function checks
00041 if the search string contains illegal characters or if it's already
00042 defined. In this cases nothing happens and an error message gets
00043 returned. Otherwise the search string and its replacement gets stored
00044 in the container.
00045
00046 The search string must not contain characters in the range CHR(0) to
00047 CHR(2).
00048
00049 '/
00050 FUNCTION RepData.add(BYREF S AS STRING, BYREF R AS STRING) AS ZSTRING PTR
00051   IF S = "" THEN RETURN 0
00052   IF INSTR(S, ANY !"\000\001\002") THEN RETURN @"undefined char (search)"
00053   VAR c = CHR(1) & S & CHR(2) : IF INSTR(5, I, c) THEN RETURN @"already defined"
00054   *CAST(INTEGER PTR, SADD(I)) += 1 : I &= MID(c, 2) & HEX(LEN(O) + 4) & CHR(1)
00055   O &= MKI(0) & R & CHR(0) : RETURN 0
00056 END FUNCTION
00057
00058 /'* \brief Search for a string, get its replacement (if any)
00059 \param S The string to search for
00060 \returns A pointer to the replacement (if any)
00061
00062 This function searches for the string given as parameter. If the string
00063 is in the search words, a pointer to it's replacement gets returned.
00064 Otherwise a pointer to the original string gets returned.
00065
00066 The function searches case sensitive.
00067
00068 '/
00069 FUNCTION RepData.rep(BYREF S AS STRING) AS ZSTRING PTR
00070   VAR a = INSTR(I, CHR(1) & S & CHR(2)) : IF a THEN a += LEN(S) + 2 ELSE RETURN SADD(S)
00071   VAR e = INSTR(a, I, CHR(1)) + 1
00072   DIM AS ZSTRING PTR z = SADD(O) + VALINT("&h" & MID(I, a, e - a))
00073   *CAST(INTEGER PTR, z - 4) += 1 : RETURN z
00074 END FUNCTION
00075
00076
00077 /'* \brief Replace special characters for HTML output
00078 \param T The Buffer to read from
00079 \param A Start position (zero based)
00080 \param L Length of substring
00081 \returns A string with replaced special characters
00082
00083 The function is used as \ref Highlighter::special_chars() function. It
00084 extracts a substring from the input buffer. Special characters are
00085 replaced by their HTML equivalents.
00086
00087 '/
00088 FUNCTION html_specials(BYVAL T AS UBYTE PTR, BYVAL A AS INTEGER, BYVAL L AS INTEGER) AS STRING
00089   STATIC AS STRING r
00090   r = ""
00091   FOR i AS INTEGER = A TO A + L - 1
00092     SELECT CASE AS CONST T[i]
00093     CASE ASC("&")  : r &= "&amp;"
00094     CASE ASC("<")  : r &= "&lt;"
00095     CASE ASC(">")  : r &= "&gt;"
00096     CASE ASC("""") : r &= "&quot;"
00097     CASE ELSE : r &= CHR(T[i])
00098     END SELECT
00099   NEXT
00100   RETURN r
00101 END FUNCTION
00102
00103 /'* \brief Replace special characters for LaTeX output
00104 \param T The Buffer to read from
00105 \param A Start position (zero based)
00106 \param L Length of substring
00107 \returns A string with replaced special characters
00108
00109 The function is used as \ref Highlighter::special_chars() function. It
00110 extracts a substring from the input buffer. Special characters are
00111 replaced by their LaTeX equivalents.
00112
00113 '/
00114 FUNCTION tex_specials(BYVAL T AS UBYTE PTR, BYVAL A AS INTEGER, BYVAL L AS INTEGER) AS STRING
00115   STATIC AS STRING r
00116   r = ""
00117   FOR i AS INTEGER = A TO A + L - 1
00118     SELECT CASE AS CONST T[i]
00119     CASE ASC("\") : r &= "\(\backslash\)"
00120     CASE ASC("#") : r &= "\#"
00121     CASE ASC("%") : r &= "\%"
00122     CASE ASC("_") : r &= "\_"
00123     CASE ASC("{") : r &= "\{"
```

```
00124     CASE ASC("}") : r &= "\}"
00125     CASE ELSE : r &= CHR(T[i])
00126     END SELECT
00127   NEXT
00128   RETURN r
00129 END FUNCTION
00130
00131 /'* \brief Replace special characters for XML output
00132 \param T The Buffer to read from
00133 \param A Start position (zero based)
00134 \param L Length of substring
00135 \returns A string with replaced special characters
00136
00137 The function is used as \ref Highlighter::special_chars() function. It
00138 extracts a substring from the input buffer. Special characters are
00139 replaced by their XML equivalents.
00140
00141 '/
00142 FUNCTION xml_specials(BYVAL T AS UBYTE PTR, BYVAL A AS INTEGER, BYVAL L AS INTEGER) AS STRING
00143   STATIC AS STRING r
00144   r = ""
00145   FOR i AS INTEGER = A TO A + L - 1
00146     SELECT CASE AS CONST T[i]
00147     CASE ASC("&")  : r &= "&amp;"
00148     CASE ASC("<")  : r &= "&lt;"
00149     CASE ASC(">")  : r &= "&gt;"
00150     CASE ASC("""") : r &= "&quot;"
00151     CASE ASC("'")  : r &= "&apos;"
00152     CASE ASC(" ")  : r &= "<sp/>"
00153     CASE ELSE : r &= CHR(T[i])
00154     END SELECT
00155   NEXT
00156   RETURN r
00157 END FUNCTION
00158
00159
00160 /'* \brief Generate end of line code for HTML output
00161 \param Symb Symbol table for cross-referencing
00162 \param Nr Line number
00163 \returns A string to end the current line and start a new one
00164
00165 The function is used as \ref Highlighter::eol() function. It generates
00166 code to end the current line and start a new one with the next line
00167 number. The number counter gets increased and returned as a BYREF
00168 parameter. Special line no. are
00169
00170 - 1: the first line (emit no line end but the line start)
00171 - 0: the last line (emit the line end but no line start)
00172
00173 '/
00174 FUNCTION html_eol(BYVAL Symb AS RepData PTR, BYREF Nr AS INTEGER) AS STRING
00175   IF Nr = 0 THEN RETURN "</div>"
00176   VAR r = "<div class=""line"">", nstr = RIGHT("    " & Nr, 5)
00177   IF Symb THEN
00178     r &= "<a name=""l" & RIGHT("0000" & Nr, 5) & """></a>"
00179     VAR res = Symb->rep(nstr)
00180     IF res = SADD(nstr) THEN r &= "<span class=""lineno"">" & nstr _
00181                         ELSE r &= "<span class=""lineno"">" & *res
00182   ELSE
00183     r &= "<span class=""lineno"">" & nstr
00184   END IF
00185   Nr += 1
00186   IF Nr = 2 THEN RETURN r & "</span> "
00187   RETURN !"</div>" & NL & r & "</span> "
00188 END FUNCTION
00189
00190 /'* \brief Generate end of line code for LaTeX output
00191 \param Symb Symbol table for cross-referencing
00192 \param Nr Line number
00193 \returns A string to end the current line and start a new one
00194
00195 The function is used as \ref Highlighter::eol() function. It generates
00196 code to end the current line and start a new one with the next line
00197 number. The number counter gets increased and returned as a BYREF
00198 parameter. Special line no. are
00199
00200 - 1: the first line (emit no line end but the line start)
00201 - 0: the last line (emit the line end but no line start)
00202
00203 '/
00204 FUNCTION tex_eol(BYVAL Symb AS RepData PTR, BYREF Nr AS INTEGER) AS STRING
00205   IF Nr = 0 THEN RETURN ""
00206   VAR  r = "" _
00207   , nstr = RIGHT("0000" & Nr, 5) _
00208    , res = IIF(Symb, Symb->rep(nstr), SADD(nstr))
00209   IF res = SADD(nstr) THEN r &= nstr _
00210                       ELSE r &= *res
```

```
00211   Nr += 1
00212   IF Nr = 2 THEN RETURN r & " "
00213   RETURN NL & r & " "
00214 END FUNCTION
00215
00216 /'* \brief Generate end of line code for XML output
00217 \param Symb Symbol table for cross-referencing
00218 \param Nr Line number
00219 \returns A string to end the current line and start a new one
00220
00221 The function is used as \ref Highlighter::eol() function. It generates
00222 code to end the current line and start a new one with the next line
00223 number. The number counter gets increased and returned as a BYREF
00224 parameter. Special line numbers are
00225
00226 - 1: the first line (emit no line end but the line start)
00227 - 0: the last line (emit the line end but no line start)
00228
00229 '/
00230 FUNCTION xml_eol(BYVAL Symb AS RepData PTR, BYREF Nr AS INTEGER) AS STRING
00231   IF Nr = 0 THEN RETURN "</codeline>"
00232   VAR  r = "<codeline lineno=""" _
00233   , nstr = "" & Nr _
00234    , res = IIF(Symb, Symb->rep(nstr), SADD(nstr))
00235   IF res = SADD(nstr) THEN r &= nstr &""">" _
00236                       ELSE r  = *res
00237   Nr += 1
00238   IF Nr = 2 THEN RETURN r
00239   RETURN "</codeline>" & NL & r
00240 END FUNCTION
00241
00242
00243 /'* \brief Class to process the syntax highlighting
00244
00245 The class is used to process the replacement of the Doxygen syntax
00246 highlighting for HTML, LaTeX and XML output. It contains members to
00247
00248 - scan for the files (Doxygen outputs),
00249 - to read files and extract links,
00250 - copy context to new files (header and footer)
00251 - replace original file by fixed version.
00252
00253 '/
00254 TYPE Highlighter
00255   '* \brief The high-lighting categories
00256   ENUM WordTypes
00257     FB_CODE  '*< Normal code, no high-lighting
00258     FB_KEYW  '*< A keyword
00259     FB_KWTP  '*< A keyword type
00260     FB_KWFL  '*< A flow keyword (currently not used)
00261     FB_PREP  '*< A preprocessor statement
00262     FB_SYMB  '*< A linked Symbol
00263   END ENUM
00264
00265   AS STRING _
00266       FbPath _   '*< The path to read FB code files from
00267     , FbFiles _  '*< A list of all FB file names
00268     , InPath _   '*< The path to read Doxygen files from
00269     , DoxyFiles _'*< A list of all Doxygen file names
00270     , HtmlPath _ '*< The path for html files
00271     , HtmlSuff _ '*< The filename suffix for html files
00272     , TexPath _  '*< The path for LaTeX files
00273     , XmlPath _  '*< The path for XML files
00274     , LastLine   '*< The last line red from the input file
00275   AS RepData PTR Symbols '*< The list of linked symbols
00276   AS Parser PTR Pars '*< The parser to operate with
00277   AS ZSTRING PTR _
00278       FBDOC_MARK = @"<!-- Syntax-highlighting by fb-doc -->" _ '*< Text to mark the output
00279     , KEYW_A = @"<span class=""keyword"">"             _ '*< Code to start highlighting a keyword
00280     , KWTP_A = @"<span class=""keywordtype"">"         _ '*< Code to start highlighting a keywordtype
00281     , KWFL_A = @"<span class=""keywordflow"">"         _ '*< Code to start highlighting a flow keyword (not used yet)
00282     , PREP_A = @"<span class=""preprocessor"">"        _ '*< Code to start highlighting a preprocessor statement
00283     , CMNT_A = @"<span class=""comment"">"             _ '*< Code to start highlighting a comment
00284     , SPAN_E = @"</span>"                              _ '*< Code to end highlighting
00285     , QUOT_A = @"<span class=""stringliteral"">&quot;" _ '*< Code to start highlighting a string literal
00286     , QUOT_E = @"&quot;</span>"                          '*< Code to end highlighting a string literal
00287   AS INTEGER _
00288       Ifnr _  '*< The file number for input
00289     , LineNo  '*< The current line number
00290
00291   UNION
00292     TYPE
00293       AS INTEGER _
00294         GenHtml : 1 _ '*< Flag for html output
00295       , GenTex : 1 _ '*< Flag for LaTeX output
00296       , GenXml : 1   '*< Flag for XML output
00297     END TYPE
```

```
00298     AS INTEGER GenAny '*< All output flags
00299   END UNION
00300
00301   DECLARE CONSTRUCTOR()
00302   DECLARE CONSTRUCTOR(BYVAL AS Parser PTR)
00303   DECLARE SUB doDoxy(BYREF AS STRING)
00304   DECLARE SUB do_files()
00305   DECLARE STATIC FUNCTION prepare_tex(BYVAL AS Highlighter PTR) AS STRING
00306   DECLARE STATIC FUNCTION prepare_xml(BYVAL AS Highlighter PTR) AS STRING
00307   DECLARE STATIC FUNCTION prepare_html(BYVAL AS Highlighter PTR) AS STRING
00308   DECLARE SUB generate_all(BYVAL AS ZSTRING PTR, BYVAL AS INTEGER)
00309   DECLARE FUNCTION generate_code(BYVAL AS ZSTRING PTR, BYVAL AS INTEGER, BYVAL AS INTEGER) AS STRING
00310   DECLARE FUNCTION word_type(BYREF AS STRING) AS ZSTRING PTR
00311
00312   '* \brief the function called to end a line and start a new one
00313   eol AS FUNCTION(BYVAL AS RepData PTR, BYref AS INTEGER) AS STRING _
00314     = @html_eol()
00315   '* \brief the function called to extract links from original files
00316   prepare AS FUNCTION(BYVAL AS Highlighter PTR) AS STRING _
00317     = @prepare_html()
00318   '* \brief the function called for normal code to replace special characters
00319   special_chars AS FUNCTION(BYVAL AS UBYTE PTR, BYVAL AS INTEGER, BYVAL AS INTEGER) AS STRING _
00320     = @html_specials()
00321 END TYPE
00322
00323 /'* \brief Constructor executing the complete process
00324 \param P The parser for input
00325
00326 This constructor connects to the parser in use.
00327
00328 '/
00329 CONSTRUCTOR Highlighter(BYVAL P AS Parser PTR)
00330   Pars = P
00331   Pars->UserTok = CAST(INTEGER PTR, @THIS)
00332 END CONSTRUCTOR
00333
00334
00335 /'* \brief Constructor executing the complete process
00336 \param Fnam The Doxyfile for input
00337
00338 This SUB controls the complete repairing process
00339
00340 - load and parse Doxyfile for paths and output types
00341 - scan FB directory for file names
00342 - switch to matching emitter parameters
00343 - scan output directories and process files
00344
00345 '/
00346 SUB Highlighter.doDoxy(BYREF Fnam AS STRING)
00347   VAR path = OPT->addPath(OPT->StartPath, LEFT(Fnam, INSTRREV(Fnam, SLASH))) _
00348     , doxy = NEW Doxyfile(Fnam) _
00349     , recu = OPT->InRecursiv _
00350     , tree = OPT->InTree
00351
00352   MSG_LINE("Doxyfile " & Fnam)
00353   WHILE doxy->Length
00354     GenHtml = IIF(doxy->Tag("GENERATE_HTML") = "YES" ANDALSO _
00355                   doxy->Tag("SOURCE_BROWSER") = "YES", 1, 0)
00356     GenTex  = IIF(doxy->Tag("GENERATE_LATEX") = "YES" ANDALSO _
00357                   doxy->Tag("LATEX_SOURCE_CODE") = "YES", 1, 0)
00358     GenXml  = IIF(doxy->Tag("GENERATE_XML") = "YES" ANDALSO _
00359                   doxy->Tag("XML_PROGRAMLISTING") = "YES", 1, 0)
00360
00361     IF GenAny THEN MSG_END("scanned") _
00362               ELSE MSG_END("nothing to do") : EXIT WHILE
00363
00364     FbPath = doxy->Tag("INPUT")
00365     OPT->InRecursiv = IIF(doxy->Tag("RECURSIVE") = "YES", 1, 0)
00366     MSG_LINE("FB source " & FbPath)
00367     CHDIR(path)
00368     IF CHDIR(FbPath) THEN MSG_END("error (couldn't change directory)") : EXIT WHILE
00369     FbFiles = NL & OPT->scanFiles("*.bas", "") _
00370                  & OPT->scanFiles("*.bi", "")
00371     IF LEN(FbFiles) > 1 THEN MSG_END("scanned") _
00372                         ELSE MSG_END("error (no FB source files)") : EXIT WHILE
00373     FbPath = OPT->addPath(path, FbPath)
00374
00375     OPT->InTree = 0
00376     InPath = doxy->Tag("OUTPUT_DIRECTORY")
00377     IF GenHtml THEN
00378       HtmlPath = OPT->addPath(InPath, doxy->Tag("HTML_OUTPUT"))
00379       HtmlSuff = doxy->Tag("HTML_FILE_EXTENSION")
00380       IF 0 = LEN(HtmlSuff) THEN HtmlSuff = ".html"
00381       OPT->InRecursiv = IIF(doxy->Tag("CREATE_SUBDIRS") = "YES", 1, 0)
00382       CHDIR(path)
00383       MSG_LINE("HTML source " & LEFT(HtmlPath, LEN(HtmlPath) - 1))
00384       IF CHDIR(HtmlPath) THEN
```

```
00385         MSG_END("error (couldn't change directory)")
00386       ELSE
00387         DoxyFiles = OPT->scanFiles("*_8bas_source" & HtmlSuff, "") _
00388                   & OPT->scanFiles("*_8bi_source" & HtmlSuff, "")
00389
00390         IF LEN(DoxyFiles) > 1 THEN MSG_END("scanned") : do_files() _
00391                              ELSE MSG_END("scanned (no files)")
00392       END IF
00393     END IF
00394     OPT->InRecursiv = 0
00395     IF GenTex THEN
00396       TexPath = OPT->addPath(InPath, doxy->Tag("LATEX_OUTPUT"))
00397       MSG_LINE("LaTeX source " & LEFT(TexPath, LEN(TexPath) - 1))
00398       CHDIR(path)
00399       IF CHDIR(TexPath) THEN
00400         MSG_END("error (couldn't change directory)")
00401       ELSE
00402         DoxyFiles = OPT->scanFiles("*_8bas_source.tex", "") _
00403                   & OPT->scanFiles("*_8bi_source.tex", "")
00404         IF LEN(DoxyFiles) > 1 THEN
00405           MSG_END("scanned")
00406           FBDOC_MARK = @"%%% Syntax-highlighting by fb-doc %%%"
00407           KEYW_A = @"\textcolor{keyword}{"
00408           KWTP_A = @"\textcolor{keywordtype}{"
00409           KWFL_A = @"\textcolor{keywordflow}{"
00410           PREP_A = @"\textcolor{preprocessor}{"
00411           CMNT_A = @"\textcolor{comment}{"
00412           SPAN_E = @"}"
00413           QUOT_A = @"\textcolor{stringliteral}{"""
00414           QUOT_E = @"""}"
00415           eol = @tex_eol()
00416           special_chars = @tex_specials()
00417           prepare = @prepare_tex()
00418           do_files()
00419         ELSE
00420           MSG_END("scanned (no files)")
00421         END IF
00422       END IF
00423     END IF
00424     IF GenXml THEN
00425       XmlPath = OPT->addPath(InPath, doxy->Tag("XML_OUTPUT"))
00426       MSG_LINE("XML source " & LEFT(XmlPath, LEN(XmlPath) - 1))
00427       CHDIR(path)
00428       IF CHDIR(XmlPath) THEN
00429         MSG_END("error (couldn't change directory)")
00430       ELSE
00431         DoxyFiles = OPT->scanFiles("*_8bas.xml", "") _
00432                   & OPT->scanFiles("*_8bi.xml", "")
00433         IF LEN(DoxyFiles) > 1 THEN
00434           MSG_END("scanned")
00435           FBDOC_MARK = @"<!-- Syntax-highlighting by fb-doc -->"
00436           KEYW_A = @"<highlight class=""keyword"">"
00437           KWTP_A = @"<highlight class=""keywordtype"">"
00438           KWFL_A = @"<highlight class=""keywordflow"">"
00439           PREP_A = @"<highlight class=""preprocessor"">"
00440           CMNT_A = @"<highlight class=""comment"">"
00441           SPAN_E = @"</highlight>"
00442           QUOT_A = @"<highlight class=""stringliteral"">"""
00443           QUOT_E = @"""</highlight>"
00444           eol = @xml_eol()
00445           special_chars = @xml_specials()
00446           prepare = @prepare_xml()
00447           do_files()
00448         ELSE
00449           MSG_END("scanned (no files)")
00450         END IF
00451       END IF
00452     END IF
00453     EXIT WHILE
00454   WEND
00455   DELETE doxy
00456   OPT->InTree = tree
00457   OPT->InRecursiv = recu
00458   CHDIR(OPT->StartPath)
00459 END SUB
00460
00461
00462 /'* \brief Operate on all files
00463
00464 The procedure operates on all file names specified in \ref DoxyFiles. It
00465 opens the input file, generated by Doxygen, writes a new file to copy
00466 the original context to (file start and end) and starts the emitter to
00467 replaces the source code section with advanced syntax highlighting.
00468
00469 Each file can only be fixed once (because the link texts from the
00470 original files are used and they change during the operation).
00471
```

```
00472 When done, the original file is killed and replaced by the new file,
00473 which is renamed to the original file name.
00474
00475 '/
00476 SUB Highlighter.do_files()
00477   VAR a = 1, e = a, l = LEN(DoxyFiles)
00478   WHILE a < l
00479     e = INSTR(a + 1, DoxyFiles, !"\n")
00480     VAR in_fnam = MID(DoxyFiles, a, e - a)
00481     Symbols = NEW RepData
00482     MSG_LINE(in_fnam)
00483     OPT->Ocha = FREEFILE
00484     IF OPEN(in_fnam & "_" FOR OUTPUT AS OPT->Ocha) THEN
00485       MSG_END("error (couldn't write)")
00486     ELSE
00487       Ifnr = FREEFILE
00488       IF OPEN(in_fnam FOR INPUT AS Ifnr) THEN
00489         MSG_END("error (couldn't read)")
00490         CLOSE #OPT->Ocha
00491         KILL(in_fnam & "_")
00492       ELSE
00493         VAR fb_nam = prepare(@THIS)
00494         IF LEN(fb_nam) THEN
00495           Pars->File_(FbPath & fb_nam, 0)
00496
00497           PRINT #OPT->Ocha, LastLine
00498           WHILE NOT EOF(Ifnr)
00499             LINE INPUT #Ifnr, LastLine
00500             PRINT #OPT->Ocha, LastLine
00501           WEND
00502           CLOSE #Ifnr
00503           CLOSE #OPT->Ocha
00504           KILL(in_fnam)
00505           NAME(in_fnam & "_", in_fnam)
00506           MSG_END(Pars->ErrMsg)
00507         ELSE
00508           CLOSE #Ifnr
00509           CLOSE #OPT->Ocha
00510           KILL(in_fnam & "_")
00511           IF LastLine = *FBDOC_MARK THEN
00512             MSG_END("error (couldn't operate twice)")
00513           ELSE
00514             MSG_END("error (incompatible format)")
00515           END IF
00516         END IF
00517       END IF
00518     END IF
00519     DELETE Symbols
00520     a = e + 1
00521   WEND
00522 END SUB
00523
00524 /'* \brief Emit source code with syntax highlighting
00525 \param Buf The buffer to read from
00526 \param Stop_ The position to stop at
00527
00528 This procedure parses a source code section, starting at \ref
00529 Parser::SrcBgn up to the given parameter Stop_. It orperates on all
00530 kind of code (strings, comments and normal code). The output gets
00531 written to the \ref Options::Ocha file.
00532
00533 '/
00534 SUB Highlighter.generate_all(BYVAL Buf AS ZSTRING PTR, BYVAL Stop_ AS INTEGER)
00535   WITH *Pars
00536     VAR i = .SrcBgn, start = i
00537     WHILE i <= Stop_
00538       SELECT CASE AS CONST Buf[i]
00539       CASE 0 : EXIT WHILE
00540       CASE ASC(!"\n")
00541         IF i <= start THEN Code(eol(Symbols, LineNo)) : start = i + 1 : EXIT SELECT
00542         Code(generate_code(Buf, start, i - start) & eol(Symbols, LineNo))
00543         start = i + 1
00544       CASE ASC("""")
00545         IF i >= start THEN Code(generate_code(Buf, start, i - start))
00546         start = i
00547         SCAN_QUOTE(Buf,i)
00548         Code(*QUOT_A & special_chars(Buf, start + 1, i - start - 1) & *QUOT_E)
00549         start = i + 1
00550       CASE ASC("'")
00551         IF i >= start THEN Code(generate_code(Buf, start, i - start))
00552         start = i
00553         SCAN_SL_COMM(Buf,i)
00554         Code(*CMNT_A & special_chars(Buf, start, i - start) & *SPAN_E)
00555         start = i
00556         CONTINUE WHILE
00557       CASE ASC("/") : IF Buf[i + 1] <> ASC("'") THEN EXIT SELECT
00558         i += 2
```

```
00559         start = i - 2
00560         DO
00561           SELECT CASE AS CONST Buf[i]
00562           CASE 0 : EXIT DO
00563           CASE ASC(!"\n")
00564             IF i <= start THEN Code(eol(Symbols, LineNo)) : start = i + 1 : EXIT SELECT
00565             Code(*CMNT_A & special_chars(Buf, start, i - start) & *SPAN_E & eol(Symbols, LineNo))
00566             start = i + 1
00567           CASE ASC("'")
00568             SELECT CASE AS CONST Buf[i + 1]
00569             CASE 0 : EXIT while
00570             CASE ASC("/") : i += 1 : EXIT DO
00571             END SELECT
00572           END SELECT : i += 1
00573         LOOP
00574
00575         Code(*CMNT_A & special_chars(Buf, start, i - start + 1) & *SPAN_E)
00576         DO
00577           SELECT CASE AS CONST Buf[i]
00578           CASE 0 : EXIT while
00579           CASE ASC(!"\r")
00580           CASE ASC(!"\n") : Code(eol(Symbols, LineNo))
00581           CASE ELSE : EXIT DO
00582           END SELECT : i += 1
00583         LOOP : start = i + 1
00584       END SELECT : i += 1
00585     WEND : IF .SrcBgn < i THEN .SrcBgn = i
00586   END WITH
00587 END SUB
00588
00589
00590 /'* \brief Check type of word
00591 \param W The word to check
00592 \returns The type of the word (first byte) & formated word
00593
00594 Check a word in the source context and return its type and a string in
00595 camel-case letters for formating in mixed cases. The first byte of this
00596 string is the word type (1 = keyword, 2 = preprocessor).
00597
00598 '/
00599 FUNCTION Highlighter.word_type(BYREF W AS STRING) AS ZSTRING PTR
00600   SELECT CASE AS CONST W[0]
00601     CASE ASC("_")
00602       SELECT CASE W
00603         CASE               "__DATE__" : RETURN @!"\004__Date__"
00604         CASE          "__DATE_ISO__" : RETURN @!"\004__Date_Iso__"
00605         CASE          "__FB_64BIT__" : RETURN @!"\004__Fb_64Bit__"
00606         CASE           "__FB_ARGC__" : RETURN @!"\004__Fb_Argc__"
00607         CASE           "__FB_ARGV__" : RETURN @!"\004__Fb_Argv__"
00608         CASE        "__FB_BACKEND__" : RETURN @!"\004__Fb_Backend__"
00609         CASE      "__FB_BIGENDIAN__" : RETURN @!"\004__Fb_Bigendian__"
00610         CASE     "__FB_BUILD_DATE__" : RETURN @!"\004__Fb_Build_Date__"
00611         CASE         "__FB_CYGWIN__" : RETURN @!"\004__Fb_Cygwin__"
00612         CASE         "__FB_DARWIN__" : RETURN @!"\004__Fb_Darwin__"
00613         CASE          "__FB_DEBUG__" : RETURN @!"\004__Fb_Debug__"
00614         CASE            "__FB_DOS__" : RETURN @!"\004__Fb_Dos__"
00615         CASE            "__FB_ERR__" : RETURN @!"\004__Fb_Err__"
00616         CASE         "__FB_FPMODE__" : RETURN @!"\004__Fb_Fpmode__"
00617         CASE            "__FB_FPU__" : RETURN @!"\004__Fb_Fpu__"
00618         CASE        "__FB_FREEBSD__" : RETURN @!"\004__Fb_Freebsd__"
00619         CASE          "__FB_LANG__" : RETURN @!"\004__Fb_Lang__"
00620         CASE          "__FB_LINUX__" : RETURN @!"\004__Fb_Linux__"
00621         CASE           "__FB_MAIN__" : RETURN @!"\004__Fb_Main__"
00622       "__FB_MIN_VERSION__" : RETURN @!"\004__Fb_Min_Version__"
00623         CASE             "__FB_MT__" : RETURN @!"\004__Fb_Mt__"
00624         CASE         "__FB_NETBSD__" : RETURN @!"\004__Fb_Netbsd__"
00625         CASE        "__FB_OPENBSD__" : RETURN @!"\004__Fb_Openbsd__"
00626       "__FB_OPTION_BYVAL__" : RETURN @!"\004__Fb_Option_Byval__"
00627     "__FB_OPTION_DYNAMIC__" : RETURN @!"\004__Fb_Option_Dynamic__"
00628     "__FB_OPTION_ESCAPE__" : RETURN @!"\004__Fb_Option_Escape__"
00629   "__FB_OPTION_EXPLICIT__" : RETURN @!"\004__Fb_Option_Explicit__"
00630       "__FB_OPTION_GOSUB__" : RETURN @!"\004__Fb_Option_Gosub__"
00631     "__FB_OPTION_PRIVATE__" : RETURN @!"\004__Fb_Option_Private__"
00632         CASE         "__FB_OUT_DLL__" : RETURN @!"\004__Fb_Out_Dll__"
00633         CASE         "__FB_OUT_EXE__" : RETURN @!"\004__Fb_Out_Exe__"
00634         CASE         "__FB_OUT_LIB__" : RETURN @!"\004__Fb_Out_Lib__"
00635         CASE         "__FB_OUT_OBJ__" : RETURN @!"\004__Fb_Out_Obj__"
00636         CASE           "__FB_PCOS__" : RETURN @!"\004__Fb_Pcos__"
00637         CASE       "__FB_SIGNATURE__" : RETURN @!"\004__Fb_Signature__"
00638         CASE            "__FB_SSE__" : RETURN @!"\004__Fb_Sse__"
00639         CASE           "__FB_UNIX__" : RETURN @!"\004__Fb_Unix__"
00640         CASE       "__FB_VECTORIZE__" : RETURN @!"\004__Fb_Vectorize__"
00641         CASE       "__FB_VER_MAJOR__" : RETURN @!"\004__Fb_Ver_Major__"
00642         CASE       "__FB_VER_MINOR__" : RETURN @!"\004__Fb_Ver_Minor__"
00643         CASE       "__FB_VER_PATCH__" : RETURN @!"\004__Fb_Ver_Patch__"
00644         CASE         "__FB_VERSION__" : RETURN @!"\004__Fb_Version__"
00645         CASE           "__FB_WIN32__" : RETURN @!"\004__Fb_Win32__"
```

```
00646         CASE            "__FB_XBOX__" : RETURN @!"\004__Fb_Xbox__"
00647         CASE               "__FILE__" : RETURN @!"\004__File__"
00648         CASE             "__FILE_NQ__" : RETURN @!"\004__File_Nq__"
00649         CASE           "__FUNCTION__" : RETURN @!"\004__Function__"
00650         CASE        "__FUNCTION_NQ__" : RETURN @!"\004__Function_Nq__"
00651         CASE               "__LINE__" : RETURN @!"\004__Line__"
00652         CASE               "__PATH__" : RETURN @!"\004__Path__"
00653         CASE               "__TIME__" : RETURN @!"\004__Time__"
00654       END SELECT
00655     CASE ASC("#")
00656       SELECT CASE W
00657         CASE   "#ASSERT" : RETURN @!"\004#Assert"
00658         CASE      "#DEF" : RETURN @!"\004#Def"
00659         CASE   "#DEFINE" : RETURN @!"\004#Define"
00660         CASE     "#ELSE" : RETURN @!"\004#Else"
00661         CASE   "#ELSEIF" : RETURN @!"\004#ElseIf"
00662         CASE    "#ENDIF" : RETURN @!"\004#EndIf"
00663         CASE "#ENDMACRO" : RETURN @!"\004#EndMacro"
00664         CASE    "#ERROR" : RETURN @!"\004#Error"
00665         CASE       "#IF" : RETURN @!"\004#If"
00666         CASE    "#IFDEF" : RETURN @!"\004#IfDef"
00667         CASE   "#IFNDEF" : RETURN @!"\004#IfNDef"
00668         CASE  "#INCLUDE" : RETURN @!"\004#Include"
00669         CASE   "#INCLIB" : RETURN @!"\004#IncLib"
00670         CASE     "#LANG" : RETURN @!"\004#Lang"
00671         CASE  "#LIBPATH" : RETURN @!"\004#LibPath"
00672         CASE     "#LINE" : RETURN @!"\004#Line"
00673         CASE    "#MACRO" : RETURN @!"\004#Macro"
00674         CASE   "#PRAGMA" : RETURN @!"\004#Pragma"
00675         CASE    "#PRINT" : RETURN @!"\004#Print"
00676         CASE    "#UNDEF" : RETURN @!"\004#UnDef"
00677       END SELECT
00678     'CASE ASC("$") ' I don't wonna test this old stuff
00679       'SELECT CASE W
00680         'CASE "$INCLUDE" : RETURN @!"\004$Include"
00681         'CASE "$DYNAMIC" : RETURN @!"\004$Dynamic"
00682         'CASE    "$LANG" : RETURN @!"\004$Lang"
00683         'CASE  "$STATIC" : RETURN @!"\004$Static"
00684       'END SELECT
00685     CASE ASC("A")
00686       SELECT CASE W
00687         CASE        "ABS" : RETURN @!"\001Abs"
00688         CASE   "ABSTRACT" : RETURN @!"\001Abstract"
00689         CASE     "ACCESS" : RETURN @!"\001Access"
00690         CASE       "ACOS" : RETURN @!"\001Acos"
00691         CASE        "ADD" : RETURN @!"\001Add"
00692         CASE      "ALIAS" : RETURN @!"\001Alias"
00693         CASE   "ALLOCATE" : RETURN @!"\001Allocate"
00694         CASE      "ALPHA" : RETURN @!"\001Alpha"
00695         CASE        "AND" : RETURN @!"\001And"
00696         CASE    "ANDALSO" : RETURN @!"\001AndAlso"
00697         CASE        "ANY" : RETURN @!"\001Any"
00698         CASE     "APPEND" : RETURN @!"\001Append"
00699         CASE         "AS" : RETURN @!"\002As"
00700         CASE        "ASC" : RETURN @!"\001Asc"
00701         CASE       "ASIN" : RETURN @!"\001Asin"
00702         CASE        "ASM" : RETURN @!"\001Asm"
00703         CASE     "ASSERT" : RETURN @!"\001Assert"
00704         CASE "ASSERTWARN" : RETURN @!"\001Assertwarn"
00705         CASE      "ATAN2" : RETURN @!"\001Atan2"
00706         CASE        "ATN" : RETURN @!"\001Atn"
00707       END SELECT
00708     CASE ASC("B")
00709       SELECT CASE W
00710         CASE     "BASE" : RETURN @!"\001Base"
00711         CASE     "BEEP" : RETURN @!"\001Beep"
00712         CASE      "BIN" : RETURN @!"\001Bin"
00713         CASE   "BINARY" : RETURN @!"\001Binary"
00714         CASE      "BIT" : RETURN @!"\001Bit"
00715         CASE "BITRESET" : RETURN @!"\001BitReset"
00716         CASE   "BITSET" : RETURN @!"\001BitSet"
00717         CASE    "BLOAD" : RETURN @!"\001Bload"
00718         CASE    "BSAVE" : RETURN @!"\001BSave"
00719         CASE     "BYTE" : RETURN @!"\002Byte"
00720         CASE    "BYREF" : RETURN @!"\002ByRef"
00721         CASE    "BYVAL" : RETURN @!"\002ByVal"
00722       END SELECT
00723     CASE ASC("C")
00724       SELECT CASE W
00725         CASE          "CALL" : RETURN @!"\001Call"
00726         CASE     "CALLOCATE" : RETURN @!"\001CAllocate"
00727         CASE         "CALLS" : RETURN @!"\001Calls"
00728         CASE          "CASE" : RETURN @!"\001Case"
00729         CASE          "CAST" : RETURN @!"\001Cast"
00730         CASE         "CBYTE" : RETURN @!"\001CByte"
00731         CASE          "CDBL" : RETURN @!"\001CDbl"
00732         CASE         "CDECL" : RETURN @!"\001Cdecl"
```

```
00733         CASE          "CHAIN" : RETURN @!"\001Chain"
00734         CASE          "CHDIR" : RETURN @!"\001ChDir"
00735         CASE            "CHR" : RETURN @!"\001Chr"
00736         CASE           "CINT" : RETURN @!"\001CInt"
00737         CASE         "CIRCLE" : RETURN @!"\001Circle"
00738         CASE          "CLASS" : RETURN @!"\001Class"
00739         CASE          "CLEAR" : RETURN @!"\001Clear"
00740         CASE           "CLNG" : RETURN @!"\001CLng"
00741         CASE        "CLNGINT" : RETURN @!"\001CLngInt"
00742         CASE          "CLOSE" : RETURN @!"\001Close"
00743         CASE            "CLS" : RETURN @!"\001Cls"
00744         CASE          "COLOR" : RETURN @!"\001Color"
00745         CASE            "COM" : RETURN @!"\001Com"
00746         CASE        "COMMAND" : RETURN @!"\001Command"
00747         CASE         "COMMON" : RETURN @!"\002Common"
00748         CASE "CONDBROADCAST" : RETURN @!"\001CondBroadcast"
00749         CASE    "CONDCREATE" : RETURN @!"\001CondCreate"
00750         CASE   "CONDDESTROY" : RETURN @!"\001CondDestroy"
00751         CASE    "CONDSIGNAL" : RETURN @!"\001CondSignal"
00752         CASE      "CONDWAIT" : RETURN @!"\001CondWait"
00753         CASE          "CONS" : RETURN @!"\001Cons"
00754         CASE         "CONST" : RETURN @!"\002Const"
00755         CASE   "CONSTRUCTOR" : RETURN @!"\001Constructor"
00756         CASE      "CONTINUE" : RETURN @!"\001Continue"
00757         CASE           "COS" : RETURN @!"\001Cos"
00758         CASE          "CPTR" : RETURN @!"\001CPtr"
00759         CASE        "CSHORT" : RETURN @!"\001CShort"
00760         CASE          "CSIGN" : RETURN @!"\001DSign"
00761         CASE           "CSNG" : RETURN @!"\001CSng"
00762         CASE        "CSRLIN" : RETURN @!"\001CsrLin"
00763         CASE        "CUBYTE" : RETURN @!"\001CUbyte"
00764         CASE          "CUINT" : RETURN @!"\001CUInt"
00765         CASE          "CULNG" : RETURN @!"\001CULng"
00766         CASE      "CULNGINT" : RETURN @!"\001CULngInt"
00767         CASE         "CUSNG" : RETURN @!"\001CUSng"
00768         CASE        "CURDIR" : RETURN @!"\001Curdir"
00769         CASE       "CUSHORT" : RETURN @!"\001CUShort"
00770         CASE        "CUSTOM" : RETURN @!"\001Custom"
00771         CASE           "CVD" : RETURN @!"\001Cvd"
00772         CASE           "CVI" : RETURN @!"\001Cvi"
00773         CASE           "CVL" : RETURN @!"\001CVL"
00774         CASE     "CVLONGINT" : RETURN @!"\001CVLongInt"
00775         CASE           "CVS" : RETURN @!"\001Cvs"
00776         CASE       "CVSHORT" : RETURN @!"\001CVShort"
00777       END SELECT
00778     CASE ASC("D")
00779       SELECT CASE W
00780         CASE         "DATA" : RETURN @!"\001Data"
00781         CASE         "DATE" : RETURN @!"\001Date"
00782         CASE      "DATEADD" : RETURN @!"\001DateAdd"
00783         CASE     "DATEDIFF" : RETURN @!"\001DateDiff"
00784         CASE     "DATEPART" : RETURN @!"\001DatePart"
00785         CASE   "DATESERIAL" : RETURN @!"\001DateSerial"
00786         CASE    "DATEVALUE" : RETURN @!"\001DateValue"
00787         CASE          "DAY" : RETURN @!"\001Day"
00788         CASE   "DEALLOCATE" : RETURN @!"\001DeAllocate"
00789         CASE      "DECLARE" : RETURN @!"\001Declare"
00790         CASE      "DEFBYTE" : RETURN @!"\001DefByte"
00791         CASE       "DEFDBL" : RETURN @!"\001DefDbl"
00792         CASE      "DEFINED" : RETURN @!"\004Defined"
00793         CASE       "DEFINT" : RETURN @!"\001DefInt"
00794         CASE       "DEFLNG" : RETURN @!"\001DefLng"
00795         CASE   "DEFLONGINT" : RETURN @!"\001DefLongInt"
00796         CASE     "DEFSHORT" : RETURN @!"\001DefShort"
00797         CASE       "DEFSNG" : RETURN @!"\001DefSng"
00798         CASE       "DEFSTR" : RETURN @!"\001DefStr"
00799         CASE     "DEFUBYTE" : RETURN @!"\001DefUbyte"
00800         CASE      "DEFUINT" : RETURN @!"\001DefUInt"
00801         CASE "DEFULONGINT" : RETURN @!"\001DefULongInt"
00802         CASE    "DEFUSHORT" : RETURN @!"\001DefUShort"
00803         CASE       "DELETE" : RETURN @!"\001Delete"
00804         CASE   "DESTRUCTOR" : RETURN @!"\001Destructor"
00805         CASE          "DIM" : RETURN @!"\002Dim"
00806         CASE          "DIR" : RETURN @!"\001Dir"
00807         CASE           "DO" : RETURN @!"\001Do"
00808         CASE       "DOUBLE" : RETURN @!"\002Double"
00809         CASE         "DRAW" : RETURN @!"\001Draw"
00810         CASE    "DYLIBFREE" : RETURN @!"\001DyLibFree"
00811         CASE    "DYLIBLOAD" : RETURN @!"\001DyLibLoad"
00812         CASE "DYLIBSYMBOL" : RETURN @!"\001DyLibSymbol"
00813         CASE      "DYNAMIC" : RETURN @!"\001Dynamic"
00814       END SELECT
00815     CASE ASC("E")
00816       SELECT CASE W
00817         CASE      "ELSE" : RETURN @!"\001Else"
00818         CASE    "ELSEIF" : RETURN @!"\001ElseIf"
00819         CASE "ENCODING" : RETURN @!"\001Encoding"
```

```
00820         CASE      "END" : RETURN @!"\001End"
00821         CASE    "ENDIF" : RETURN @!"\001EndIf"
00822         CASE     "ENUM" : RETURN @!"\001Enum"
00823         CASE  "ENVIRON" : RETURN @!"\001Environ"
00824         CASE      "EOF" : RETURN @!"\001Eof"
00825         CASE      "EQV" : RETURN @!"\001Eqv"
00826         CASE    "ERASE" : RETURN @!"\001Erase"
00827         CASE     "ERFN" : RETURN @!"\001Erfn"
00828         CASE      "ERL" : RETURN @!"\001Erl"
00829         CASE     "ERMN" : RETURN @!"\001Ermn"
00830         CASE      "ERR" : RETURN @!"\001Err"
00831         CASE    "ERROR" : RETURN @!"\001Error"
00832         CASE   "ESCAPE" : RETURN @!"\001Escape"
00833         CASE     "EXEC" : RETURN @!"\001Exec"
00834         CASE  "EXEPATH" : RETURN @!"\001ExePath"
00835         CASE "EXPLICIT" : RETURN @!"\001Explicit"
00836         CASE     "EXIT" : RETURN @!"\001Exit"
00837         CASE      "EXP" : RETURN @!"\001Exp"
00838         CASE   "EXPORT" : RETURN @!"\001Export"
00839         CASE  "EXTENDS" : RETURN @!"\001Extends"
00840         CASE   "EXTERN" : RETURN @!"\002Extern"
00841         CASE   "ESCAPE" : RETURN @!"\001Escape"
00842         CASE "EXPLICIT" : RETURN @!"\001Explicit"
00843       END SELECT
00844     CASE ASC("F")
00845       SELECT CASE W
00846         CASE        "FIELD" : RETURN @!"\001Field"
00847         CASE     "FILEATTR" : RETURN @!"\001FileAttr"
00848         CASE     "FILECOPY" : RETURN @!"\001FileCopy"
00849         CASE "FILEDATETIME" : RETURN @!"\001FileDateTime"
00850         CASE   "FILEEXISTS" : RETURN @!"\001FileExists"
00851         CASE      "FILELEN" : RETURN @!"\001FileLen"
00852         CASE          "FIX" : RETURN @!"\001Fix"
00853         CASE         "FLIP" : RETURN @!"\001Flip"
00854         CASE          "FOR" : RETURN @!"\001For"
00855         CASE       "FORMAT" : RETURN @!"\001Format"
00856         CASE         "FRAC" : RETURN @!"\001Frac"
00857         CASE          "FRE" : RETURN @!"\001Fre"
00858         CASE     "FREEFILE" : RETURN @!"\001FreeFile"
00859         CASE     "FUNCTION" : RETURN @!"\001Function"
00860       END SELECT
00861     CASE ASC("G")
00862       SELECT CASE W
00863         CASE         "GET" : RETURN @!"\001Get"
00864         CASE "GETJOYSTICK" : RETURN @!"\001GetJoyStick"
00865         CASE      "GETKEY" : RETURN @!"\001GetKey"
00866         CASE    "GETMOUSE" : RETURN @!"\001GetMouse"
00867         CASE       "GOSUB" : RETURN @!"\001GoSub"
00868         CASE        "GOTO" : RETURN @!"\001Goto"
00869       END SELECT
00870     CASE ASC("H")
00871       SELECT CASE W
00872         CASE    "HEX" : RETURN @!"\001Hex"
00873         CASE "HIBYTE" : RETURN @!"\001HiByte"
00874         CASE "HIWORD" : RETURN @!"\001HiWord"
00875         CASE   "HOUR" : RETURN @!"\001Hour"
00876       END SELECT
00877     CASE ASC("I")
00878       SELECT CASE W
00879         CASE             "IF" : RETURN @!"\001If"
00880         CASE            "IIF" : RETURN @!"\001IIf"
00881         CASE "IMAGECONVERTROW" : RETURN @!"\001ImageConvertRow"
00882         CASE     "IMAGECREATE" : RETURN @!"\001ImageCreate"
00883         CASE    "IMAGEDESTROY" : RETURN @!"\001ImageDestroy"
00884         CASE       "IMAGEINFO" : RETURN @!"\001ImageInfo"
00885         CASE             "IMP" : RETURN @!"\001Imp"
00886         CASE      "IMPLEMENTS" : RETURN @!"\001Implements"
00887         CASE          "IMPORT" : RETURN @!"\001Import"
00888         CASE           "INKEY" : RETURN @!"\001InKey"
00889         CASE             "INP" : RETURN @!"\001Inp"
00890         CASE           "INPUT" : RETURN @!"\001Input"
00891         CASE           "INSTR" : RETURN @!"\001Instr"
00892         CASE        "INSTRREV" : RETURN @!"\001InstrRev"
00893         CASE             "INT" : RETURN @!"\001Int"
00894         CASE         "INTEGER" : RETURN @!"\002Integer"
00895         CASE              "IS" : RETURN @!"\001Is"
00896         CASE          "ISDATE" : RETURN @!"\001IsDate"
00897       END SELECT
00898     CASE ASC("K")
00899       SELECT CASE W
00900         CASE "KILL" : RETURN @!"\001Kill"
00901       END SELECT
00902     CASE ASC("L")
00903       SELECT CASE W
00904         CASE  "LBOUND" : RETURN @!"\001LBound"
00905         CASE   "LCASE" : RETURN @!"\001LCase"
00906         CASE    "LEFT" : RETURN @!"\001Left"
```

```
00907         CASE     "LEN" : RETURN @!"\001Len"
00908         CASE     "LET" : RETURN @!"\001Let"
00909         CASE     "LIB" : RETURN @!"\001Lib"
00910         CASE    "LINE" : RETURN @!"\001Line"
00911         CASE  "LOBYTE" : RETURN @!"\001LoByte"
00912         CASE     "LOC" : RETURN @!"\001Loc"
00913         CASE   "LOCAL" : RETURN @!"\001Local"
00914         CASE  "LOCATE" : RETURN @!"\001Locate"
00915         CASE    "LOCK" : RETURN @!"\001Lock"
00916         CASE     "LOF" : RETURN @!"\001Lof"
00917         CASE     "LOG" : RETURN @!"\001Log"
00918         CASE    "LONG" : RETURN @!"\002Long"
00919         CASE "LONGINT" : RETURN @!"\002LongInt"
00920         CASE    "LOOP" : RETURN @!"\001Loop"
00921         CASE  "LOWORD" : RETURN @!"\001LoWord"
00922         CASE    "LPOS" : RETURN @!"\001LPos"
00923         CASE  "LPRINT" : RETURN @!"\001LPrint"
00924         CASE     "LPT" : RETURN @!"\001Lpt"
00925         CASE    "LSET" : RETURN @!"\001LSet"
00926         CASE   "LTRIM" : RETURN @!"\001LTrim"
00927       END SELECT
00928     CASE ASC("M")
00929       SELECT CASE W
00930         CASE          "MID" : RETURN @!"\001Mid"
00931         CASE       "MINUTE" : RETURN @!"\001Minute"
00932         CASE          "MKD" : RETURN @!"\001MKD"
00933         CASE        "MKDIR" : RETURN @!"\001MkDir"
00934         CASE          "MKI" : RETURN @!"\001MKi"
00935         CASE          "MKL" : RETURN @!"\001MKl"
00936         CASE    "MKLONGINT" : RETURN @!"\001MKLongInt"
00937         CASE          "MKS" : RETURN @!"\001MKs"
00938         CASE      "MKSHORT" : RETURN @!"\001MKShort"
00939         CASE          "MOD" : RETURN @!"\001Mod"
00940         CASE        "MONTH" : RETURN @!"\001Month"
00941         CASE    "MONTHNAME" : RETURN @!"\001MonthName"
00942         CASE     "MULTIKEY" : RETURN @!"\001MultiKey"
00943         CASE  "MUTEXCREATE" : RETURN @!"\001MutexCreate"
00944         CASE "MUTEXDESTROY" : RETURN @!"\001MutexDestroy"
00945         CASE    "MUTEXLOCK" : RETURN @!"\001MutexLock"
00946         CASE  "MUTEXUNLOCK" : RETURN @!"\001MutexUnlock"
00947       END SELECT
00948     CASE ASC("N")
00949       SELECT CASE W
00950         CASE     "NAKED" : RETURN @!"\001Naked"
00951         CASE      "NAME" : RETURN @!"\001Name"
00952         CASE "NAMESPACE" : RETURN @!"\001Namespace"
00953         CASE       "NEW" : RETURN @!"\001New"
00954         CASE      "NEXT" : RETURN @!"\001Next"
00955         CASE       "NOT" : RETURN @!"\001Not"
00956         CASE       "NOW" : RETURN @!"\001Now"
00957         CASE   "NOGOSUB" : RETURN @!"\001NoGoSub"
00958         CASE "NOKEYWORD" : RETURN @!"\001NoKeyWord"
00959       END SELECT
00960     CASE ASC("O")
00961       SELECT CASE W
00962         CASE   "OBJECT" : RETURN @!"\001Object"
00963         CASE      "OCT" : RETURN @!"\001Oct"
00964         CASE "OFFSETOF" : RETURN @!"\001OffsetOf"
00965         CASE       "ON" : RETURN @!"\001On"
00966         CASE     "ONCE" : RETURN @!"\004Once"
00967         CASE     "OPEN" : RETURN @!"\001Open"
00968         CASE "OPERATOR" : RETURN @!"\001Operator"
00969         CASE   "OPTION" : RETURN @!"\001Option"
00970         CASE       "OR" : RETURN @!"\001Or"
00971         CASE   "ORELSE" : RETURN @!"\001OrElse"
00972         CASE      "OUT" : RETURN @!"\001Out"
00973         CASE   "OUTPUT" : RETURN @!"\001Output"
00974         CASE "OVERLOAD" : RETURN @!"\001Overload"
00975         CASE "OVERRIDE" : RETURN @!"\001Override"
00976         CASE   "OPTION" : RETURN @!"\001Option"
00977       END SELECT
00978     CASE ASC("P")
00979       SELECT CASE W
00980         CASE     "PAINT" : RETURN @!"\001Paint"
00981         CASE   "PALETTE" : RETURN @!"\001Palette"
00982         CASE    "PASCAL" : RETURN @!"\001Pascal"
00983         CASE     "PCOPY" : RETURN @!"\001PCopy"
00984         CASE      "PEEK" : RETURN @!"\001Peek"
00985         CASE      "PIPE" : RETURN @!"\001Pipe"
00986         CASE      "PMAP" : RETURN @!"\001Pmap"
00987         CASE     "POINT" : RETURN @!"\001Point"
00988         CASE   "POINTER" : RETURN @!"\001Pointer"
00989         CASE      "POKE" : RETURN @!"\001Poke"
00990         CASE       "POS" : RETURN @!"\001Pos"
00991         CASE       "POP" : RETURN @!"\001Pop"
00992         CASE  "PRESERVE" : RETURN @!"\001Preserve"
00993         CASE    "PRESET" : RETURN @!"\001Preset"
```

```
00994         CASE     "PRINT" : RETURN @!"\001Print"
00995         CASE   "PRIVATE" : RETURN @!"\001Private"
00996         CASE   "PROCPTR" : RETURN @!"\001ProcPtr"
00997         CASE  "PROPERTY" : RETURN @!"\001Property"
00998         CASE "PROTECTED" : RETURN @!"\001Protected"
00999         CASE      "PSET" : RETURN @!"\001PSet"
01000         CASE       "PTR" : RETURN @!"\002Ptr"
01001         CASE    "PUBLIC" : RETURN @!"\001Public"
01002         CASE       "PUT" : RETURN @!"\001Put"
01003         CASE      "PUSH" : RETURN @!"\001Push"
01004       END SELECT
01005     CASE ASC("R")
01006       SELECT CASE W
01007         CASE     "RANDOM" : RETURN @!"\001Random"
01008         CASE  "RANDOMIZE" : RETURN @!"\001Randomize"
01009         CASE       "READ" : RETURN @!"\001Read"
01010         CASE "REALLOCATE" : RETURN @!"\001ReAllocate"
01011         CASE      "REDIM" : RETURN @!"\001ReDim"
01012         CASE        "REM" : RETURN @!"\001Rem"
01013         CASE      "RESET" : RETURN @!"\001Reset"
01014         CASE    "RESTORE" : RETURN @!"\001Restore"
01015         CASE     "RESUME" : RETURN @!"\001Resume"
01016         CASE     "RETURN" : RETURN @!"\001Return"
01017         CASE        "RGB" : RETURN @!"\001RGB"
01018         CASE       "RGBA" : RETURN @!"\001RGBA"
01019         CASE      "RIGHT" : RETURN @!"\001Right"
01020         CASE      "RMDIR" : RETURN @!"\001RmDir"
01021         CASE        "RND" : RETURN @!"\001Rnd"
01022         CASE       "RSET" : RETURN @!"\001RSet"
01023         CASE      "RTRIM" : RETURN @!"\001RTrim"
01024         CASE        "RUN" : RETURN @!"\001Run"
01025       END SELECT
01026     CASE ASC("S")
01027       SELECT CASE W
01028         CASE          "SADD" : RETURN @!"\001SAdd"
01029         CASE         "SCOPE" : RETURN @!"\001Scope"
01030         CASE        "SCREEN" : RETURN @!"\001Screen"
01031         CASE "SCREENCONTROL" : RETURN @!"\001ScreenControl"
01032         CASE    "SCREENCOPY" : RETURN @!"\001ScreenCopy"
01033         CASE   "SCREENEVENT" : RETURN @!"\001ScreenEvent"
01034         CASE  "SCREENGLPROC" : RETURN @!"\001ScreenGlProc"
01035         CASE    "SCREENINFO" : RETURN @!"\001ScreenInfo"
01036         CASE    "SCREENLIST" : RETURN @!"\001ScreenList"
01037         CASE    "SCREENLOCK" : RETURN @!"\001ScreenLock"
01038         CASE     "SCREENPTR" : RETURN @!"\001ScreenPtr"
01039         CASE     "SCREENRES" : RETURN @!"\001ScreenRes"
01040         CASE     "SCREENSET" : RETURN @!"\001ScreenSet"
01041         CASE    "SCREENSYNC" : RETURN @!"\001ScreenSync"
01042         CASE  "SCREENUNLOCK" : RETURN @!"\001ScreenUnlock"
01043         CASE          "SCRN" : RETURN @!"\001Scrn"
01044         CASE        "SECOND" : RETURN @!"\001Second"
01045         CASE          "SEEK" : RETURN @!"\001Seek"
01046         CASE        "SELECT" : RETURN @!"\001Select"
01047         CASE       "SETDATE" : RETURN @!"\001SetDate"
01048         CASE    "SETENVIRON" : RETURN @!"\001SetEnviron"
01049         CASE      "SETMOUSE" : RETURN @!"\001SetMouse"
01050         CASE       "SETTIME" : RETURN @!"\001SetTime"
01051         CASE           "SGN" : RETURN @!"\001Sgn"
01052         CASE        "SHARED" : RETURN @!"\002Shared"
01053         CASE         "SHELL" : RETURN @!"\001Shell"
01054         CASE           "SHL" : RETURN @!"\001Shl"
01055         CASE         "SHORT" : RETURN @!"\002Short"
01056         CASE           "SHR" : RETURN @!"\001Shr"
01057         CASE           "SIN" : RETURN @!"\001Sin"
01058         CASE        "SINGLE" : RETURN @!"\002Single"
01059         CASE        "SIZEOF" : RETURN @!"\001SizeOf"
01060         CASE         "SLEEP" : RETURN @!"\001Sleep"
01061         CASE         "SPACE" : RETURN @!"\001Space"
01062         CASE           "SPC" : RETURN @!"\001Spc"
01063         CASE           "SQR" : RETURN @!"\001Sqr"
01064         CASE        "STATIC" : RETURN @!"\002Static"
01065         CASE       "STDCALL" : RETURN @!"\001StdCall"
01066         CASE          "STEP" : RETURN @!"\001Step"
01067         CASE         "STICK" : RETURN @!"\001Stick"
01068         CASE          "STOP" : RETURN @!"\001Stop"
01069         CASE           "STR" : RETURN @!"\001Str"
01070         CASE         "STRIG" : RETURN @!"\001Strig"
01071         CASE        "STRING" : RETURN @!"\002String"
01072         CASE        "STRPTR" : RETURN @!"\001StrPtr"
01073         CASE           "SUB" : RETURN @!"\001Sub"
01074         CASE          "SWAP" : RETURN @!"\001Swap"
01075         CASE        "SYSTEM" : RETURN @!"\001System"
01076       END SELECT
01077     CASE ASC("T")
01078       SELECT CASE W
01079         CASE           "TAB" : RETURN @!"\001Tab"
01080         CASE           "TAN" : RETURN @!"\001Tan"
```

```
01081         CASE         "THEN" : RETURN @!"\001Then"
01082         CASE         "THIS" : RETURN @!"\001This"
01083         CASE   "THREADCALL" : RETURN @!"\001ThreadCall"
01084         CASE "THREADCREATE" : RETURN @!"\001ThreadCreate"
01085         CASE   "THREADWAIT" : RETURN @!"\001ThreadWait"
01086         CASE         "TIME" : RETURN @!"\001Time"
01087         CASE        "TIMER" : RETURN @!"\001Timer"
01088         CASE   "TIMESERIAL" : RETURN @!"\001TimeSerial"
01089         CASE    "TIMEVALUE" : RETURN @!"\001TimeValue"
01090         CASE           "TO" : RETURN @!"\001To"
01091         CASE        "TRANS" : RETURN @!"\001Trans"
01092         CASE         "TRIM" : RETURN @!"\001Trim"
01093         CASE         "TYPE" : RETURN @!"\001Type"
01094         CASE       "TYPEOF" : RETURN @!"\004TypeOf"
01095       END SELECT
01096     CASE ASC("U")
01097       SELECT CASE W
01098         CASE   "UBOUND" : RETURN @!"\001UBound"
01099         CASE    "UBYTE" : RETURN @!"\002UByte"
01100         CASE    "UCASE" : RETURN @!"\001UCase"
01101         CASE "UINTEGER" : RETURN @!"\002UInteger"
01102         CASE    "ULONG" : RETURN @!"\002ULong"
01103         CASE "ULONGINT" : RETURN @!"\002ULongInt"
01104         CASE    "UNION" : RETURN @!"\001Union"
01105         CASE   "UNLOCK" : RETURN @!"\001Unlock"
01106         CASE "UNSIGNED" : RETURN @!"\002Unsigned"
01107         CASE    "UNTIL" : RETURN @!"\001Until"
01108         CASE   "USHORT" : RETURN @!"\002UShort"
01109         CASE    "USING" : RETURN @!"\001Using"
01110       END SELECT
01111     CASE ASC("V")
01112       SELECT CASE W
01113         CASE   "VA_ARG" : RETURN @!"\001Va_Arg"
01114         CASE "VA_FIRST" : RETURN @!"\001Va_First"
01115         CASE  "VA_NEXT" : RETURN @!"\001Va_Next"
01116         CASE      "VAL" : RETURN @!"\001Val"
01117         CASE    "VAL64" : RETURN @!"\001Val64"
01118         CASE   "VALINT" : RETURN @!"\001ValInt"
01119         CASE   "VALLNG" : RETURN @!"\001ValLng"
01120         CASE  "VALUINT" : RETURN @!"\001ValUInt"
01121         CASE  "VALULNG" : RETURN @!"\001ValULng"
01122         CASE      "VAR" : RETURN @!"\002Var"
01123         CASE   "VARPTR" : RETURN @!"\001VarPtr"
01124         CASE     "VIEW" : RETURN @!"\001View"
01125         CASE  "VIRTUAL" : RETURN @!"\001Virtual"
01126       END SELECT
01127     CASE ASC("W")
01128       SELECT CASE W
01129         CASE        "WAIT" : RETURN @!"\001Wait"
01130         CASE        "WBIN" : RETURN @!"\001WBin"
01131         CASE        "WCHR" : RETURN @!"\001WChr"
01132         CASE     "WEEKDAY" : RETURN @!"\001WeekDay"
01133         CASE "WEEKDAYNAME" : RETURN @!"\001WeekDayName"
01134         CASE        "WEND" : RETURN @!"\001Wend"
01135         CASE        "WHEX" : RETURN @!"\001WHex"
01136         CASE       "WHILE" : RETURN @!"\001While"
01137         CASE       "WIDTH" : RETURN @!"\001Width"
01138         CASE      "WINDOW" : RETURN @!"\001Window"
01139         CASE "WINDOWTITLE" : RETURN @!"\001WindowTitle"
01140         CASE      "WINPUT" : RETURN @!"\001WInput"
01141         CASE        "WITH" : RETURN @!"\001With"
01142         CASE        "WOCT" : RETURN @!"\001WOct"
01143         CASE       "WRITE" : RETURN @!"\001Write"
01144         CASE      "WSPACE" : RETURN @!"\001WSpace"
01145         CASE        "WSTR" : RETURN @!"\001WStr"
01146         CASE     "WSTRING" : RETURN @!"\002WString"
01147       END SELECT
01148     CASE ASC("X")
01149       SELECT CASE W
01150         CASE     "XOR" : RETURN @!"\001Xor"
01151       END SELECT
01152     CASE ASC("Y")
01153       SELECT CASE W
01154         CASE    "YEAR" : RETURN @!"\001Year"
01155       END SELECT
01156     CASE ASC("Z")
01157       SELECT CASE W
01158         CASE "ZSTRING" : RETURN @!"\002ZString"
01159       END SELECT
01160   END SELECT : RETURN 0
01161 END FUNCTION
01162
01163 /'* \brief check a word in source code
01164
01165 This macro is used for single source reasons. It reads the current word
01166 from the input buffer and checks if it matches
01167
```

```
01168 -# an entry in the container \ref Highlighter::Symbols (links red from orig. file),
01169 -# the FB keyword list
01170
01171 The result is in the typ variable:
01172 1. = keyword,
01173 2. = keywordtype,
01174 3. = preprocessor,
01175 4. or greater = link (type gets a ZSTRING PTR to the context)
01176
01177 '/
01178 #MACRO GET_WORD_TYPE()
01179   VAR word = MID(*T, start + 1, size)
01180   VAR res = IIF(Symbols, Symbols->rep(word), SADD(word))
01181   typ = FB_CODE
01182   IF res = SADD(word) THEN '     no symbol, check keyword & preprocessor
01183     word = UCASE(word)
01184     res = word_type(word)
01185     typ = IIF(res, ASC(*res), FB_CODE)
01186
01187     SELECT CASE AS CONST OPT->CaseMode '  reformat letter cases, if required
01188     CASE OPT->CASE_LOWER : MID(*T, start + 1, size) = LCASE(word)
01189     CASE OPT->CASE_MIXED : MID(*T, start + 1, size) = MID(*res, 2)
01190     CASE OPT->CASE_UPPER : MID(*T, start + 1, size) = word
01191     END SELECT
01192   ELSE
01193     typ = CAST(INTEGER, res) '               start of replacement string
01194   END IF
01195 #ENDMACRO
01196
01197 /'* \brief highlight normal source code
01198 \param T The input buffer from the parser
01199 \param A The start of the part to operate on (zero based)
01200 \param L The length of the part to operate on
01201 \returns Formated html code for the input context
01202
01203 This function highlights a piece of code. It doesn't handle comments
01204 nor string literals. In normal code it separates keywords,
01205 preprocessors and symbols and encovers the context by the matching
01206 tags. In the the result special characters get replaced, like '&' by
01207 '&amp;', '<' by '&lt;', ... (special characters are different in HTML,
01208 LaTeX and XML output).
01209
01210 '/
01211 FUNCTION Highlighter.generate_code(BYVAL T AS ZSTRING PTR, BYVAL A AS INTEGER, BYVAL L AS INTEGER) AS STRING
01212   STATIC AS STRING r, blcks
01213   r = "" : blcks = ""
01214   VAR size = 0, start = -1, last = -1, typ = last
01215   FOR i AS INTEGER = A TO A + L - 1
01216     SELECT CASE AS CONST T[i]
01217     CASE ASC("0") TO ASC("9")
01218       IF size THEN size += 1
01219     'CASE ASC("$") '                   I don't wonna test this old stuff
01220       'if size = 0 then
01221         'size = 1 : start = i
01222       'else
01223         'IF last <> FB_CODE THEN blcks &= MKI(FB_CODE) & MKI(i) : last = FB_CODE
01224       'end if
01225     CASE ASC("#"), ASC("_"), ASC("A") TO ASC("Z"), ASC("a") TO ASC("z")
01226       size += 1 : IF size = 1 THEN start = i
01227     CASE ELSE
01228       IF size THEN
01229         GET_WORD_TYPE()
01230         IF typ <> last THEN blcks &= MKI(typ) & MKI(i - size) : last = typ
01231         size = 0
01232       ENDIF
01233
01234       SELECT CASE AS CONST T[i]
01235       CASE ASC("?")
01236         IF last = FB_KEYW THEN EXIT SELECT
01237         blcks &= MKI(FB_KEYW) & MKI(i) : last = FB_KEYW
01238       CASE ASC(!"\t"), ASC(!"\v"), ASC(" ") '                 skip these
01239         IF i > A ANDALSO typ < FB_SYMB THEN EXIT SELECT
01240         blcks &= MKI(FB_CODE) & MKI(i) : last = FB_CODE
01241       CASE ELSE
01242         IF last = FB_CODE THEN EXIT SELECT
01243         blcks &= MKI(FB_CODE) & MKI(i) : last = FB_CODE
01244       END SELECT
01245     END SELECT
01246   NEXT
01247
01248   IF size THEN
01249     GET_WORD_TYPE()
01250     IF typ <> last THEN blcks &= MKI(typ) & MKI(A + L - size)
01251   ENDIF
01252   blcks &= MKI(-1) & MKI(A + L)
01253
01254   VAR p = CAST(INTEGER PTR, SADD(blcks))
```

```
01255   FOR i AS INTEGER = 0 TO LEN(blcks) SHR 2 - 3 STEP 2
01256     VAR l = p[i + 3] - p[i + 1]
01257     IF l THEN
01258       SELECT CASE AS CONST p[i]
01259       CASE FB_KEYW : r &= *KEYW_A & special_chars(T, p[i + 1], l) & *SPAN_E
01260       CASE FB_KWTP : r &= *KWTP_A & special_chars(T, p[i + 1], l) & *SPAN_E
01261       CASE FB_KWFL : r &= *KWFL_A & special_chars(T, p[i + 1], l) & *SPAN_E
01262       CASE FB_PREP : r &= *PREP_A & special_chars(T, p[i + 1], l) & *SPAN_E
01263       CASE FB_CODE : r &=           special_chars(T, p[i + 1], l)
01264       CASE ELSE    : r &= PEEK(ZSTRING, p[i])
01265       END SELECT
01266     END IF
01267   NEXT
01268   RETURN r
01269 END FUNCTION
01270
01271 /'* \brief Prepare a HTML file for syntax repairing
01272 \param Hgh The Highlighter to operate with
01273 \returns The file name of the FB source code
01274
01275 This function prepares a HTML file to replace the syntax highlighting.
01276 It reads the header from the original output and copies the context to
01277 the replacement file. The file name of the odiginal FB source is
01278 extracted from the header. Then the links from the original listing
01279 part are extracted in to the \ref Highlighter::Symbols table.
01280
01281 '/
01282 FUNCTION Highlighter.prepare_html(BYVAL Hgh AS Highlighter PTR) AS STRING
01283   WITH PEEK(Highlighter, Hgh)
01284     VAR pa = 0, fb_nam = ""
01285     WHILE NOT EOF(.Ifnr) '                  search start of code section
01286       LINE INPUT #.Ifnr, .LastLine
01287       pa = INSTR(.LastLine, "<div class=""line""><a name=""100001""") : IF pa THEN EXIT WHILE
01288       Code(.LastLine & NL)
01289       pa = INSTR(.LastLine, "<div class=""title"">") ' extract file name
01290       IF pa THEN
01291         pa += 19
01292         VAR pe = INSTR(pa, .LastLine, "</div>")
01293         VAR p = INSTR(.FbFiles, MID(.LastLine, pa, pe - pa)) : IF 0 = p THEN RETURN ""
01294         DO '                        search path and name of FB source list
01295           SELECT CASE AS CONST .FbFiles[p - 2]
01296           CASE ASC(NL) '                                  in main folder
01297             fb_nam = MID(.LastLine, pa, pe - pa)
01298             EXIT DO
01299           CASE ASC(SLASH) '                                 in subfolder
01300             VAR pp = INSTRREV(.FbFiles, NL, p) + 1
01301             fb_nam = MID(.FbFiles, pp, p - pp + pe - pa)
01302             EXIT DO
01303           CASE ELSE '                   only partial match, search again
01304             p = INSTR(p + 1, .FbFiles, MID(.LastLine, pa, pe - pa))
01305             IF p < 1 THEN pa = 0 : RETURN "" '     no matching FB source
01306           END SELECT
01307         LOOP
01308       END IF
01309     WEND
01310
01311     IF pa <= 1 THEN .LastLine = *.FBDOC_MARK : RETURN "" '  already done
01312     Code(LEFT(.LastLine, pa - 1) & NL)
01313     Code(*.FBDOC_MARK & NL)
01314
01315     WHILE NOT EOF(.Ifnr) '                              search for links
01316       pa = INSTR(pa + 1, .LastLine, "<a class=""code""")
01317       WHILE pa '                        extract all links from this line
01318         VAR pe = INSTR(pa + 14, .LastLine, "</a>"), pp = INSTRREV(.LastLine, ">", pe - 1) + 1
01319         VAR word = MID(.LastLine, pp, pe - pp)
01320         VAR res = .Symbols->add(word, MID(.LastLine, pa, pe - pa + 4))
01321         pa = INSTR(pa + 1, .LastLine, "<a class=""code""")
01322       WEND
01323       LINE INPUT #.Ifnr, .LastLine
01324       IF LEFT(.LastLine, 6) = "</div>" THEN EXIT WHILE
01325     WEND
01326     IF EOF(.Ifnr) THEN RETURN "" ELSE RETURN fb_nam
01327   END WITH
01328 END FUNCTION
01329
01330 /'* \brief Prepare a LaTeX file for syntax repairing
01331 \param Hgh The Highlighter to operate with
01332 \returns The file name of the FB source code
01333
01334 This function prepares a LaTeX file to replace the syntax highlighting.
01335 It reads the header from the original output and copies the context to
01336 the replacement file. The file name of the odiginal FB source is
01337 extracted from the header. Then the links from the original listing
01338 part are extracted in to the \ref Highlighter::Symbols table.
01339
01340 '/
01341 FUNCTION Highlighter.prepare_tex(BYVAL Hgh AS Highlighter PTR) AS STRING
```

```
01342   WITH PEEK(Highlighter, Hgh)
01343     LINE INPUT #.Ifnr, .LastLine '    read first line, extract file name
01344     Code(.LastLine & NL)
01345     VAR p1 = INSTR(.LastLine, "\section{") _
01346       , p2 = INSTR(p1 + 10, .LastLine, "}") _
01347       , fb_nam = ""
01348
01349     FOR i AS INTEGER = p1 + 8 TO p2 - 2 '         remove LaTeX formating
01350       IF .LastLine[i] =  ASC("\") THEN
01351         SELECT CASE AS CONST .LastLine[i + 1]
01352         CASE ASC("+"), ASC("-"), ASC("/") : i += 1 : CONTINUE FOR
01353         CASE ASC("_") : CONTINUE FOR
01354         END SELECT
01355       END IF
01356       fb_nam &= CHR(.LastLine[i])
01357     NEXT
01358
01359     WHILE NOT EOF(.Ifnr) '                  search start of code section
01360       LINE INPUT #.Ifnr, .LastLine
01361       Code(.LastLine & NL)
01362       IF .LastLine = "\begin{DoxyCode}" THEN EXIT WHILE
01363     WEND
01364
01365     IF EOF(.Ifnr) THEN RETURN ""
01366     LINE INPUT #.Ifnr, .LastLine : IF .LastLine = *.FBDOC_MARK THEN RETURN ""
01367     Code(*.FBDOC_MARK & NL)
01368
01369     DO '                                            search for links
01370       VAR pa = INSTR(.LastLine, "\hyper")
01371       IF pa = 1 THEN                              ' link for line number
01372         pa = INSTR(14, .LastLine, "\hyperlink{")
01373         pa = INSTR(pa + 10, .LastLine, "}{") + 2
01374         VAR pe = INSTR(pa, .LastLine, "}")
01375         .Symbols->add(MID(.LastLine, pa, pe - pa), LEFT(.LastLine, pe))
01376         pa = INSTR(pe + 1, .LastLine, "\hyperlink{")
01377       END IF
01378       WHILE pa '                                    links in source code
01379         VAR pm = INSTR(pa + 11, .LastLine, "}{") + 2 _
01380           , pe = INSTR(pm, .LastLine, "}") + 1 _
01381           , word = ""
01382         FOR i AS INTEGER = pm - 1 TO pe - 3
01383           IF .LastLine[i] <> ASC("\") THEN word &= CHR(.LastLine[i])
01384         NEXT
01385         .Symbols->add(word, MID(.LastLine, pa, pe - pa))
01386         pa = INSTR(pe, .LastLine, "\hyperlink{")
01387       WEND
01388       IF EOF(.Ifnr) THEN RETURN "" ELSE LINE INPUT #.Ifnr, .LastLine
01389     LOOP UNTIL .LastLine = "\end{DoxyCode}"
01390   END WITH
01391   RETURN fb_nam
01392 END FUNCTION
01393
01394 /'* \brief Prepare a XML file for syntax repairing
01395 \param Hgh The Highlighter to operate with
01396 \returns The file name of the FB source code
01397
01398 This function prepares a HTML file to replace the syntax highlighting.
01399 It reads the header from the original output and copies the context to
01400 the replacement file. The file name of the odiginal FB source is
01401 extracted from the header. Then the links from the original listing
01402 part are extracted in to the \ref Highlighter::Symbols table.
01403
01404 '/
01405 FUNCTION Highlighter.prepare_xml(BYVAL Hgh AS Highlighter PTR) AS STRING
01406   WITH PEEK(Highlighter, Hgh)
01407     VAR fb_nam = ""
01408     WHILE NOT EOF(.Ifnr) '                  search start of code section
01409       LINE INPUT #.Ifnr, .LastLine
01410       Code(.LastLine & NL)
01411       IF INSTR(.LastLine, "<programlisting>") THEN EXIT WHILE
01412       VAR pa = INSTR(.LastLine, "<compoundname>")
01413       IF pa THEN
01414         pa += 14
01415         VAR pe = INSTR(pa, .LastLine, "</compoundname>")
01416         fb_nam = MID(.LastLine, pa, pe - pa)
01417       END IF
01418     WEND
01419
01420     LINE INPUT #.Ifnr, .LastLine : IF .LastLine = *.FBDOC_MARK THEN RETURN ""
01421     Code(*.FBDOC_MARK & NL)
01422     DO
01423       VAR word = "", pm = INSTR(.LastLine, "refid=""")
01424       IF pm THEN
01425         VAR pa = INSTRREV(.LastLine, "<", pm - 1)
01426         pm = INSTR(pm + 8, .LastLine, ">")
01427         IF MID(.LastLine, pa, 5) = "<ref " THEN
01428           VAR pe = INSTR(pm + 1, .LastLine, "</ref>")
```

```
01429           pm += 1
01430           .Symbols->add(MID(.LastLine, pm, pe - pm), MID(.LastLine, pa, pe - pa + 6))
01431         ELSE
01432           VAR pe = pm + 1
01433           pm = INSTR(pa + 1, .LastLine, "lineno=""") + 8
01434           .Symbols->add(MID(.LastLine, pm, INSTR(pm + 1, .LastLine, """") - pm), _
01435                         MID(.LastLine, pa, pe - pa))
01436         END IF
01437       END IF
01438       IF EOF(.Ifnr) THEN RETURN "" ELSE LINE INPUT #.Ifnr, .LastLine
01439     LOOP UNTIL INSTR(.LastLine, "</programlisting>")
01440     IF EOF(.Ifnr) THEN RETURN "" ELSE RETURN fb_nam
01441   END WITH
01442 END FUNCTION
01443
01444
01445 /'* \brief Emitter to be called per file before parsing starts
01446 \param P the parser calling this emitter
01447
01448 This emitter gets called before the parser starts its parsing process.
01449 It initializes the FB source code emission.
01450
01451 '/
01452 SUB synt_init CDECL(BYVAL P AS Parser PTR)
01453   WITH *P
01454     IF 0 = .UserTok THEN  ' not in --syntax.mode, create new Highlighter
01455       VAR x = NEW Highlighter(P)
01456       IF 0 = OPT->InTree THEN .Po = .Fin '              skip all parsing
01457       x->LineNo = 1
01458     END IF
01459     .SrcBgn = 0
01460   END WITH
01461   WITH *CAST(Highlighter PTR, P->UserTok)
01462     IF .GenAny THEN .LineNo = 1 ELSE .Pars = P
01463     Code(.eol(.Symbols, .LineNo))
01464   END WITH
01465 END SUB
01466
01467
01468 /'* \brief Emitter to be called after parsing of a file
01469 \param P the parser calling this emitter
01470
01471 This emitter gets called after the parser ends its parsing process.
01472 It sends the rest of the FB source code to the output stream.
01473
01474 '/
01475 SUB synt_exit CDECL(BYVAL P AS Parser PTR)
01476   WITH *CAST(Highlighter PTR, P->UserTok)
01477     .generate_all(SADD(P->Buf), P->Fin)
01478     'IF OPT->Level > 0 ORELSE OPT->RunMode <> OPT->FILE_MODE THEN EXIT SUB ' we're in #INCLUDE
01479     IF OPT->Level > 0 THEN EXIT SUB ' we're in #INCLUDE
01480
01481     Code(.eol(.Symbols, 0) & NL)
01482     IF 0 = .GenAny THEN DELETE P->UserTok : P->UserTok = 0 ' not in --syntax-mode
01483   END WITH
01484 END SUB
01485
01486
01487 /'* \brief Emitter to generate an include statement
01488 \param P the parser calling this emitter
01489
01490 This emitter operates on include statements. It extracts the file name
01491 and checks the \ref Highlighter::Symbols table for a matching link. If there
01492 is no link, nothing is done.
01493
01494 In case of a matching link the source code gets emitted up to the link.
01495
01496 File names need special handling because the string literals don't get
01497 checked for linkage.
01498
01499 '/
01500 SUB synt_incl CDECL(BYVAL P AS Parser PTR)
01501   WITH *CAST(Highlighter PTR, P->UserTok)
01502     VAR fnam = TRIM(P->SubStr(P->NamTok), """") _
01503        , res = IIF(.Symbols, .Symbols->rep(fnam), SADD(fnam))
01504
01505     .generate_all(SADD(P->Buf), P->Tk1[1])
01506     VAR a = P->StaTok[1] _
01507       , l = P->NamTok[1] - a
01508
01509     Code(.generate_code(SADD(P->Buf), a - 1, l) & " """ & *res & """")
01510     P->SrcBgn = P->NamTok[1] + P->NamTok[2]
01511     IF .GenAny ORELSE 0 = OPT->InTree THEN EXIT SUB
01512
01513     Code(.eol(.Symbols, .LineNo))
01514     P->Include(fnam)
01515     .Pars = P
```

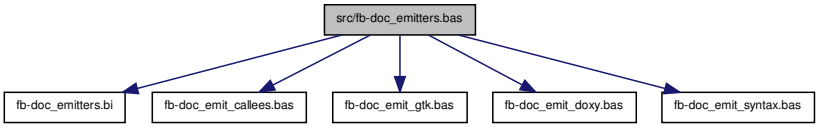
```
01516   END WITH
01517 END SUB
01518
01519
01520 /'* \brief Emitter to generate a function name
01521 \param P the parser calling this emitter
01522
01523 This emitter operates on SUB / FUNCTION / PROPERTY definitions
01524 (function body, not declaration). It extracts the function name and
01525 checks the \ref Highlighter::Symbols table for a matching link. If
01526 there is no link, nothing is done.
01527
01528 In case of a matching link the source code gets emitted up to the link.
01529
01530 Funktion names need special handling because the names of CONSTRUCTORs
01531 and DESTRUCTORs vary between intermediate format and FB source.
01532
01533 '/
01534 SUB synt_func_ CDECL(BYVAL P AS Parser PTR)
01535   WITH *CAST(Highlighter PTR, P->UserTok)
01536     VAR t = P->NamTok, nam = P->SubStr(t)
01537
01538     SELECT CASE AS CONST *P->FunTok ' create a Doxygen name to search link
01539     CASE P->TOK_CTOR : t += 3 : IF OPT->Types = OPT->FB_STYLE THEN nam &= "::" & nam
01540     CASE P->TOK_DTOR : t += 3 : IF OPT->Types = OPT->FB_STYLE THEN nam &= "::~" & nam
01541     CASE ELSE
01542       WHILE t < P->EndTok
01543         t += 3
01544         SELECT CASE AS CONST *t
01545         CASE P->TOK_DOT  : IF OPT->Types = OPT->FB_STYLE THEN nam &= "::" ELSE nam &= "."
01546         CASE P->TOK_WORD : nam &= P->SubStr(t)
01547         CASE ELSE : EXIT WHILE
01548         END SELECT
01549       WEND
01550     END SELECT
01551     nam = .special_chars(SADD(nam), 0, LEN(nam)) '      LaTeX underscore
01552     VAR res = IIF(.Symbols, .Symbols->rep(nam), SADD(nam))
01553     IF res = SADD(nam) THEN EXIT SUB
01554
01555     .generate_all(SADD(P->Buf), P->Tk1[1])
01556     VAR i = INSTR(*res, nam) _
01557       , j = i + LEN(nam) _
01558       , a = P->StaTok[1] _
01559       , l = P->NamTok[1] - a
01560     P->SrcBgn = *(t - 2) + *(t - 1)
01561     nam = .special_chars(SADD(P->Buf), a + l, P->SrcBgn - P->NamTok[1]) ' orig. name
01562     Code(.generate_code(SADD(P->Buf), a, l) _
01563         & LEFT(*res, i - 1) & nam & MID(*res, j))
01564   END WITH
01565 END SUB
01566
01567
01568 ' place the handlers in the emitter interface
01569 WITH_NEW_EMITTER("SyntaxHighLighting")
01570     .Init_ = @synt_init
01571     .Exit_ = @synt_exit
01572     .Incl_ = @synt_incl
01573     .Func_ = @synt_func_
01574 END WITH
01575
```

15.23 src/fb-doc_emitters.bas File Reference

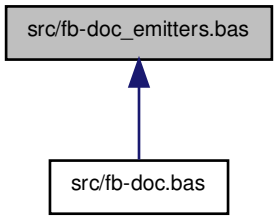
Default emitter to create pseudo C source, #INCLUDEs the other emitters.

```
#include "fb-doc_emitters.bi"
#include "fb-doc_emit_callees.bas"
#include "fb-doc_emit_gtk.bas"
#include "fb-doc_emit_doxy.bas"
#include "fb-doc_emit_syntax.bas"
```

Include dependency graph for fb-doc_emitters.bas:



This graph shows which files directly or indirectly include this file:



Functions

- SUB_CDECL cEmitComments (BYVAL_AS_Parser_PTR P, BYVAL_AS_INTEGER Stop_)

  Handler for exporting comments.
- SUB_CDECL cppNam (BYVAL_AS_Parser_PTR P)

  Export the name (including double colons for member functions)
- SUB_CDECL cNam (BYVAL_AS_Parser_PTR P)

  Export the name (including dots for member functions)
- SUB_CDECL cIni (BYVAL_AS_Parser_PTR P)

  Export the initializer(s) for a variable.
- SUB_CDECL cArrDim (BYVAL_AS_Parser_PTR P)

  The dimension of a variable.
- SUB_CDECL cppCreateTypNam (BYVAL_AS_Parser_PTR P)

  Create pseude C declaration.
- SUB_CDECL cCreateTypNam (BYVAL_AS_Parser_PTR P)

  Create C declaration.
- SUB_CDECL cppCreateFunction (BYVAL_AS_Parser_PTR P)

  Create a function declaration.
- SUB_CDECL cppEntryListParameter (BYVAL_AS_Parser_PTR P)

  Handler for a parameter declaration.
- SUB_CDECL cCreateFunction (BYVAL_AS_Parser_PTR P)

  Create a function declaration.
- SUB_CDECL cEntryListParameter (BYVAL_AS_Parser_PTR P)

  Handler for a parameter declaration.
- SUB_CDECL c_include (BYVAL_AS_Parser_PTR P)

Emitter to generate an #INCLUDE translation.

- SUB_CDECL c_defi_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a macro translation.
- SUB_CDECL c_func_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a function translation.
- SUB_CDECL c_decl_ (BYVAL_AS_Parser_PTR P)

  Emitter to generate a declaration translation.
- SUB_CDECL cEntryBlockENUM (BYVAL_AS_Parser_PTR P)

  Handler for an enumerator entry (inside ENUM block)
- SUB_CDECL c_Block (BYVAL_AS_Parser_PTR P)

  Emitter to generate a block translation.
- SUB_CDECL cEntryBlockTypeUnion (BYVAL_AS_Parser_PTR P)

  Handler for a context line (TYPE / UNION block)
- SUB_CDECL c_error (BYVAL_AS_Parser_PTR P)

  Emitter for an error message.
- SUB_CDECL c_Init (BYVAL_AS_Parser_PTR P)

  Emitter to be called before parsing.
- SUB_CDECL c_exit (BYVAL_AS_Parser_PTR P)

  Emitter to be called after parsing.
- SUB_CDECL c_CTOR (BYVAL_AS_Parser_PTR P)

  CTOR to be called when starting in Options::FileModi.
- SUB_CDECL cEmitSource (BYVAL_AS_Parser_PTR P, BYVAL_AS_INTEGER E)

  Handler for exporting FreeBasic source code.
- SUB_CDECL geanyInit (BYVAL_AS_Parser_PTR P)

  Handler for initialize the export of source code.
- SUB_CDECL geanyExit (BYVAL_AS_Parser_PTR P)

  Handler for finalize the export of source code.

Variables

- STRING LOFN

15.23.1 Detailed Description

Default emitter to create pseudo C source, #INCLUDEs the other emitters. This file is the main file for emitters. It contains some helper functions to extract original source code and comments. And it contains the standard emitter to translate the FB source to pseudo C source.

The pseudo C emitter is designed to create C source out of the FB code including the documentation comments. Comments may be multi line comment blocks or line end comments. The emitter tries to place the comments similar as in the FB source (but a comment inside a statement will be placed at the end of the corresponding C statement).

The emitter is designed to be used to generate output for the documentation tool-chain back-end. This may be written to the STDOUT pipe (default) or to one or more files (option `--fileoutout`). It also can be used to translate an FB header to C syntax by unsing option `--cstyle`.

Definition in file fb-doc_emitters.bas.

15.23.2 Function Documentation

15.23.2.1 SUB_CDECL cEmitComments (BYVAL_AS_Parser_PTR *P*, BYVAL_AS_INTEGER *Stop_*)

Handler for exporting comments.

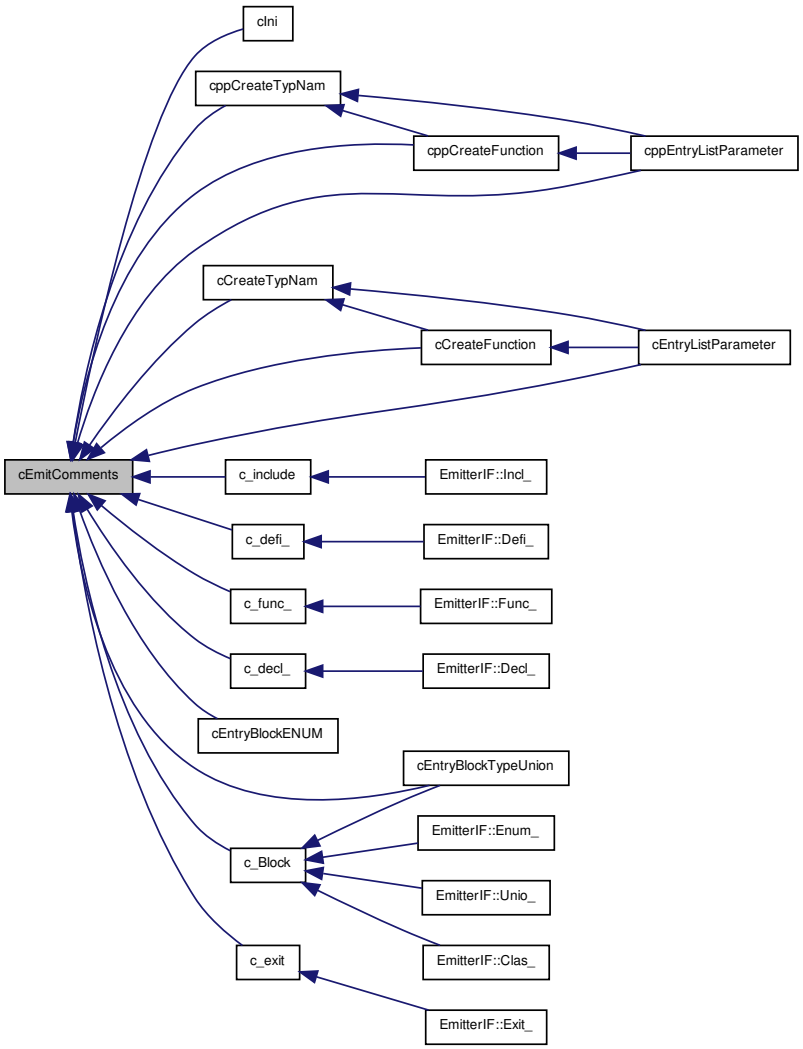
Parameters

P	the parser calling this handler
*Stop_*	the end position in the input buffer

Export the comments between the last position 'SrcBgn' (= source begin) and the Stop_ position. Then, the Stop_ position gets the new 'last position'.

Definition at line 36 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.2 SUB_CDECL cppNam (BYVAL_AS_Parser_PTR *P*)

Export the name (including double colons for member functions)

Parameters

P	the parser calling this emitter

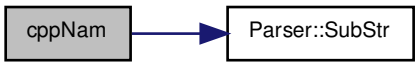
This property reads the name of a construct from the input buffer and emits all words, concatenated by double colons.

Returns

the name including dots (colons)

Definition at line 85 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.23.2.3 SUB_CDECL cNam (BYVAL_AS_Parser_PTR *P*)

Export the name (including dots for member functions)

Parameters

P	the parser calling this emitter

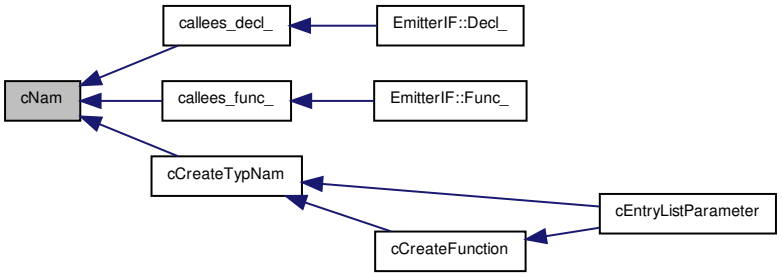
This sub reads the name of a construct from the input buffer and emits all words as in the original source.

Returns

the name including dots (colons)

Definition at line 110 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.4 SUB_CDECL clni (BYVAL_AS_Parser_PTR *P*)

Export the initializer(s) for a variable.

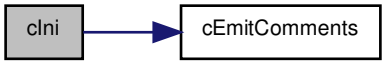
Parameters

P	the parser calling this emitter

This sub reads the initializer from the input buffer and emits it unchanged. This may be a bunch of text in case of an array initializer.

Definition at line 131 of file fb-doc_emitters.bas.

Here is the call graph for this function:



15.23.2.5 SUB_CDECL cArrDim (BYVAL_AS_Parser_PTR *P*)

The dimension of a variable.

Parameters

P	the parser calling this emitter

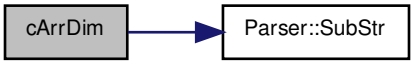
This property reads the dimension of a construct from the input buffer.

Returns

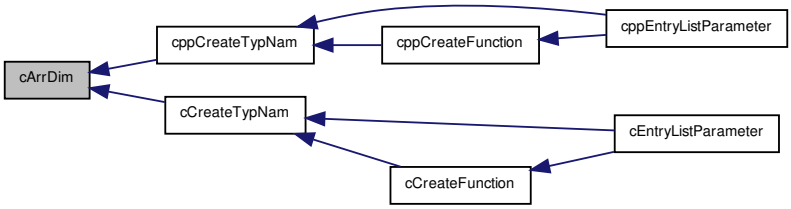
the name including round or squared brackets

Definition at line 179 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.6 SUB_CDECL cppCreateTypNam (BYVAL_AS_Parser_PTR *P*)

Create pseude C declaration.

Parameters

P	the parser calling this emitter

Create a declaration for the construct at the current parser position using FB style. All FB keywords gets mangled to a single word to get a FreeBasic look-and-feel in the documentation. Ie we emit

- "SUB Name();" (instead of "void Name(void);")
- "INTEGER varnam" (instead of "int varname")
- "BYREF_AS_STRING strng" (instead of "char∗∗ strng")
- "FUNCTION_CDECL_AS_SINGLE xyz CDECL(BYVAL_AS_BYTE C)" (instead of "float xyz(char C)")
- ...

The C lexer of the back-end (gtk-doc or Doxygen) interprets this single word as a C type declaration (or macro) and can handle FB source code that way.

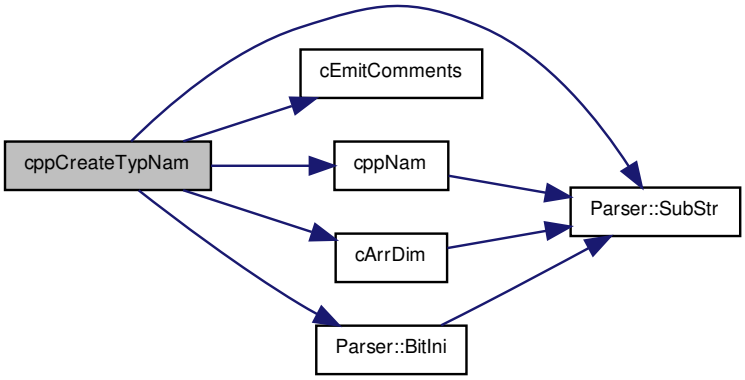
Exeptions handled in this SUB:

- TypTok = 0: no type, emit only the name (with a space in from)

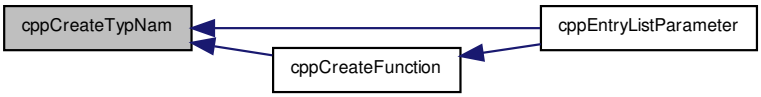
- NymTok = 0: no name, emit type and exit

Definition at line 225 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.7 SUB_CDECL cCreateTypNam (BYVAL_AS_Parser_PTR *P*)

Create C declaration.

Parameters

P	the parser calling this emitter

Create a C declaration for the construct at the current parser position. All FB keywords gets translated to their C expressions. Ie we emit

- "void Name(void);" (instead of "SUB Name();")
- "int varname" (instead of "INTEGER varnam")
- "char** strng" (instead of "BYREF_AS_STRING strng")
- "float xyz(char C)" (instead of "FUNCTION_CDECL_AS_SINGLE xyz CDECL(BYVAL_AS_BYTE C)")
- ...

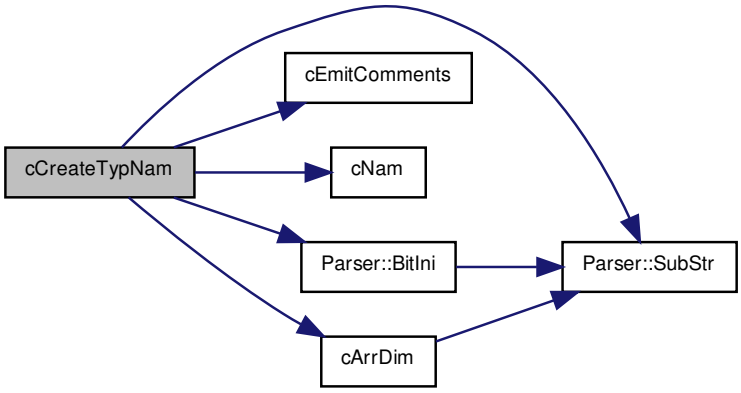
The C source code is very useful when you use a library compiled with FreeBasic in a other language like C or C++. fb-doc can auto-generate the header files (just check initializers and array dimensioning manually).

Exeptions handled in this SUB:

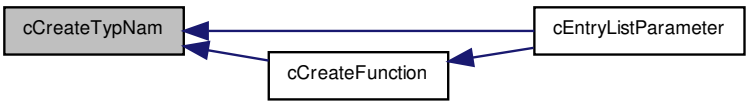
- TypTok = 0: no type, emit only the name (with a space in from)
- NymTok = 0: no name, emit type and exit

Definition at line 275 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.8 SUB_CDECL cppCreateFunction (BYVAL_AS_Parser_PTR *P*)

Create a function declaration.

Parameters

P	the parser calling this handler

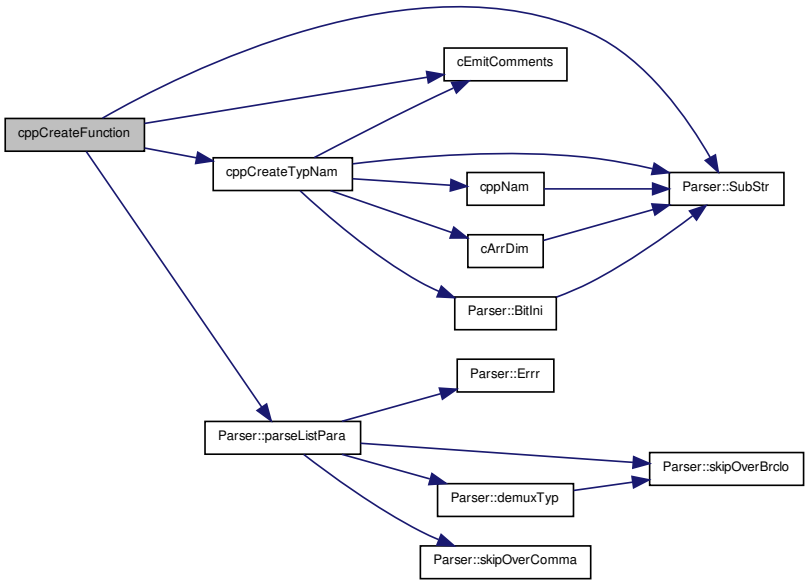
Generate a declaration for a function (SUB, FUNCTION, DESTRUCTOR, CONSTRUCTOR, PROPERTY). We emit a type, a name and a parameter list.

Exceptions:

- in a declaration a CONSTRUCTOR or DESTRUCTOR have no name

Definition at line 402 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.23.2.9 SUB_CDECL cppEntryListParameter (BYVAL_AS_Parser_PTR *P*)

Handler for a parameter declaration.

Parameters

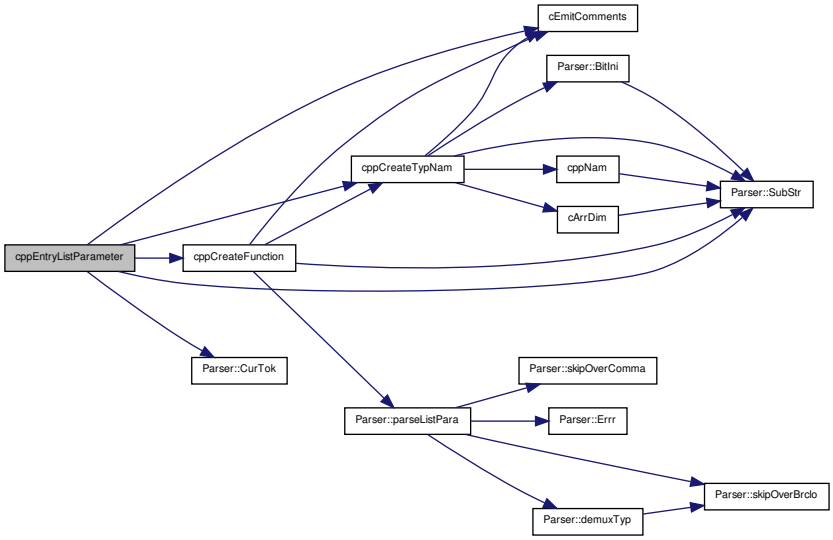
P	the parser calling this handler

Generate a declaration for a parameter list. The declaration may be empty (), may have no name (prototype declaration) or may by an ellipsis (...). Initializers get emitted "as-is".

We emit a space in front and a comma behind the parameter. When done the first space gets replaced by a '(' and the last comma gets replaced by a ')'.

Definition at line 333 of file fb-doc_emitters.bas.

Here is the call graph for this function:



15.23.2.10 SUB_CDECL cCreateFunction (BYVAL_AS_Parser_PTR *P*)

Create a function declaration.

Parameters

P	the parser calling this handler

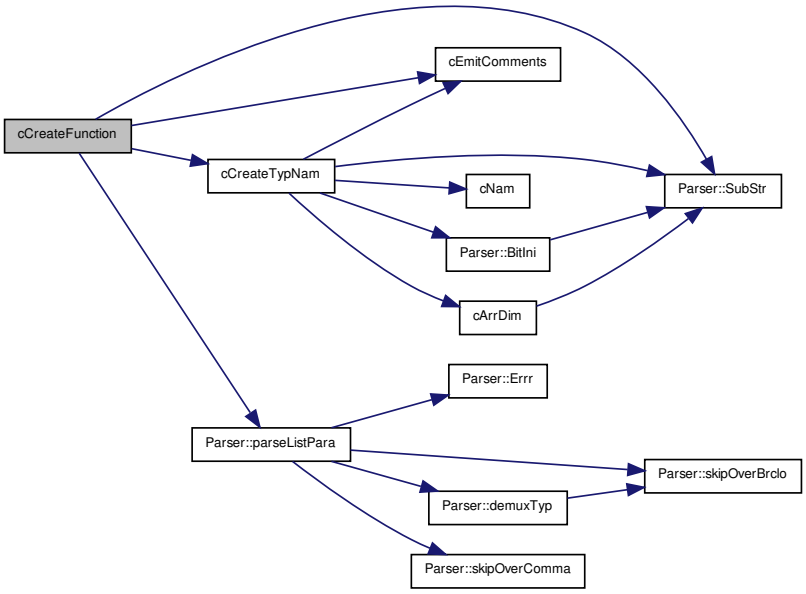
Generate a declaration for a function (SUB, FUNCTION, DESTRUCTOR, CONSTRUCTOR, PROPERTY). We emit a type, a name and the parameter list.

Exceptions:

- in a declaration a CONSTRUCTOR or DESTRUCTOR has no name. We use the block name instead.

Definition at line 442 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:

15.23.2.11 SUB_CDECL cEntryListParameter (BYVAL_AS_Parser_PTR *P*)

Handler for a parameter declaration.

Parameters

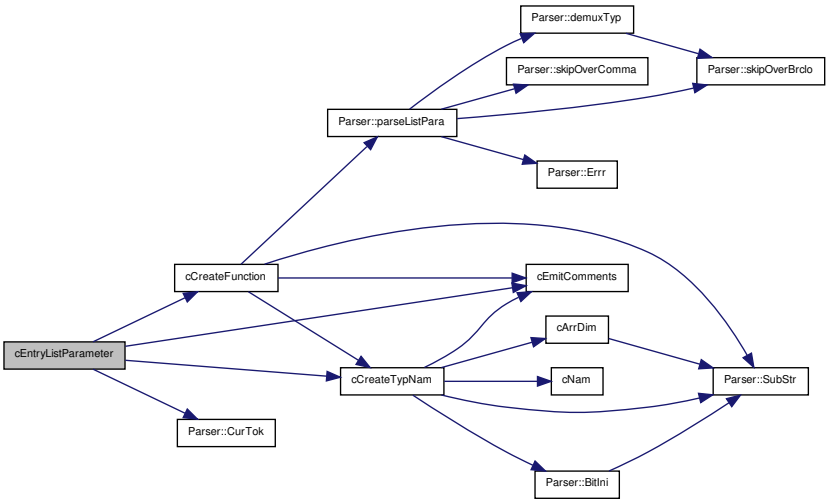
P	the parser calling this handler

Generate a declaration for a parameter list. The declaration may be empty (), may have no name (prototype declaration) or may by an ellipsis (...). Initializers get emitted "as-is", this means you have to check if they contain FB keywords (and translate manually, if so).

We emit a space in front and a comma behind the parameter. When done the first space gets replaced by a '(' and the last comma gets replaced by a ')'.

Definition at line 371 of file fb-doc_emitters.bas.

Here is the call graph for this function:



15.23.2.12 SUB_CDECL c_include (BYVAL_AS_Parser_PTR *P*)

Emitter to generate an #INCLUDE translation.

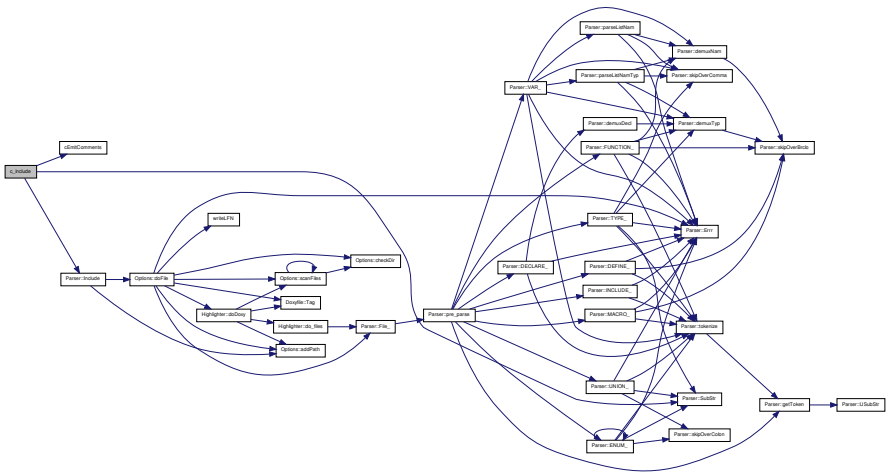
Parameters

P	the parser calling this emitter

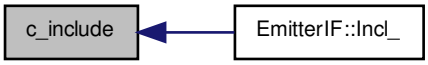
This emitter gets called when the parser finds an #INCLUDE statement. It creates a C translation and sends it to the output stream. When option --tree is given it checks if the file has been done already. If not, it creates a new Parser and starts its scanning process.

Definition at line 478 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.13 SUB_CDECL c_defi_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a macro translation.

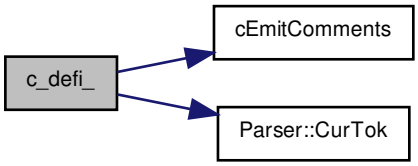
Parameters

P	the parser calling this emitter

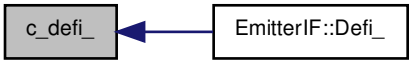
This emitter gets called when the parser finds a macro (#DEFINE #MACRO). It generates a C translation of the macro and sends it to the output stream.

Definition at line 503 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.14 SUB_CDECL c_func_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a function translation.

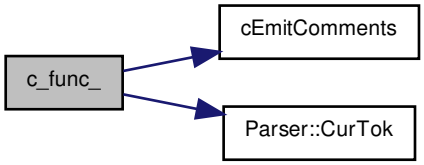
Parameters

P	the parser calling this emitter

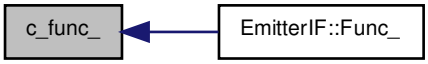
This emitter gets called when the parser finds a function (SUB / FUNCTION / PROPERTY / CONSTRUCTOR / DESTRUCTOR). It translates a function and its parameter list to C-like code and sends it to the output stream. The function body is either empty or contains pseudo calls.

Definition at line 536 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.15 SUB_CDECL c_decl_ (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a declaration translation.

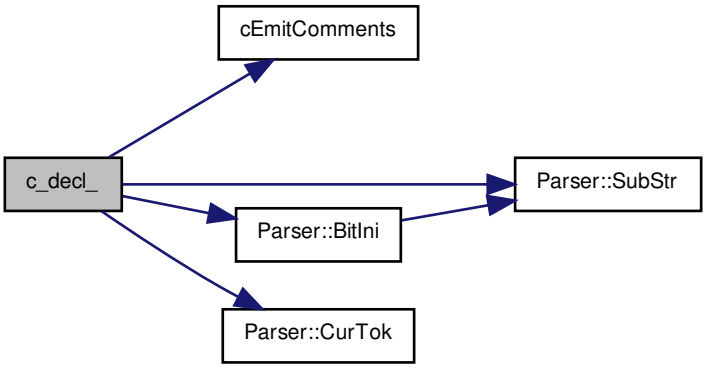
Parameters

P	the parser calling this emitter

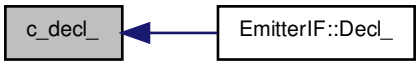
This emitter gets called when the parser is in a declaration (VAR / DIM / CONST / COMMON / EXTERN / STATIC). It generates a C translation for each variable name and sends it (them) to the output stream. Documantation comments get emitted at the appropriate place. Each declaration get a single line, even if the original source code is a comma-separated list. (This may destroy line synchonisation, so it's better to place each declaration in a single line.)

Definition at line 608 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.16 SUB_CDECL cEntryBlockENUM (BYVAL_AS_Parser_PTR *P*)

Handler for an enumerator entry (inside ENUM block)

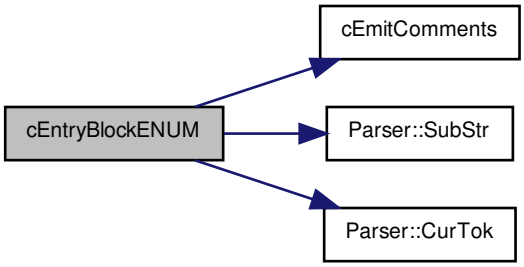
Parameters

P	the parser calling this handler

Generate an enumerator in an enum block. Name, initializers and documentation comments are emitted. Logical operators like SHL or AND are not handled jet.

Definition at line 648 of file fb-doc_emitters.bas.

Here is the call graph for this function:



15.23.2.17 SUB_CDECL c_Block (BYVAL_AS_Parser_PTR *P*)

Emitter to generate a block translation.

Parameters

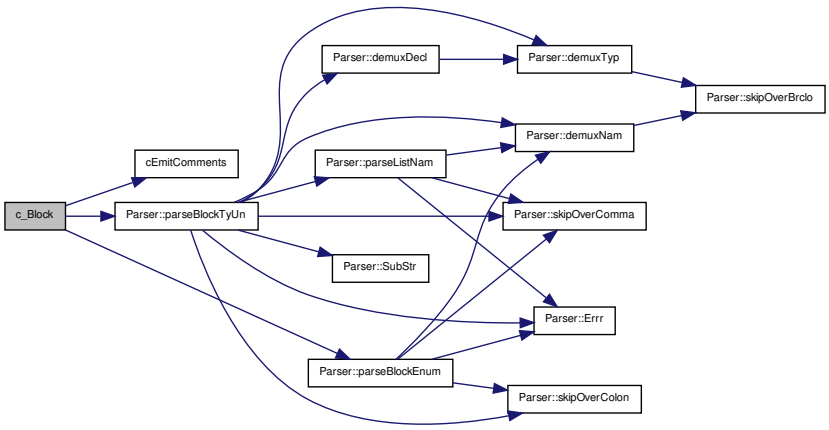
P	the parser calling this emitter

This emitter gets called when the parser finds a block (TYPE / UNION / ENUM). It generates a C translation of the block and sends it to the output stream.

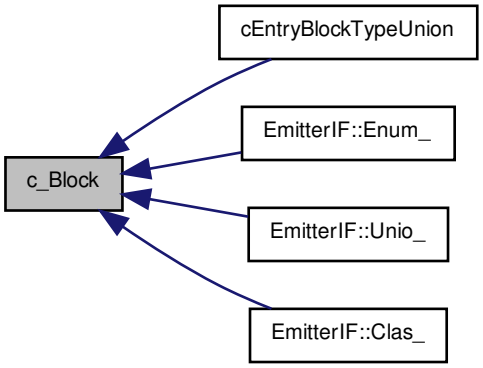
Nested blocks get parsed recursivly.

Definition at line 702 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.18 SUB_CDECL cEntryBlockTypeUnion (BYVAL_AS_Parser_PTR *P*)

Handler for a context line (TYPE / UNION block)

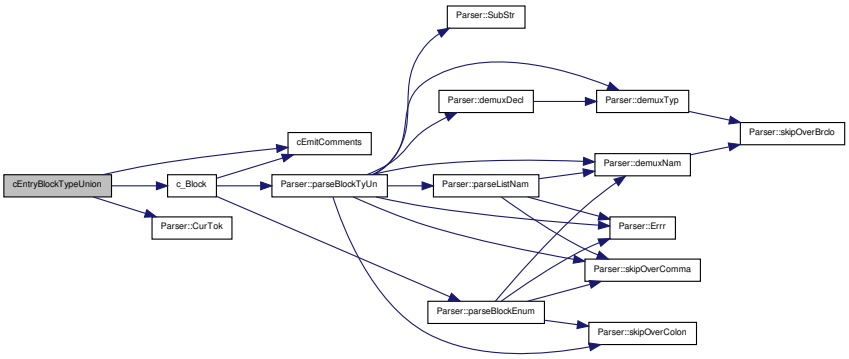
Parameters

P	the parser calling this handler

Generate a line in an struct or union block. Type, name, initializers and documentation comments are emitted. Logical operators like SHL or AND are not handled jet.

Definition at line 671 of file fb-doc_emitters.bas.

Here is the call graph for this function:



15.23.2.19 SUB_CDECL c_error (BYVAL_AS_Parser_PTR *P*)

Emitter for an error message.

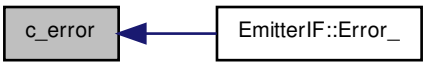
Parameters

P	the parser calling this handler

Generate an error output. When the parser detects an error it calls this function. Depending on the run-mode we do or do not emit an information. In mode `--geany-mode` the error message gets shown in the status line (or in the debug window).

Definition at line 745 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.20 SUB_CDECL c_Init (BYVAL_AS_Parser_PTR *P*)

Emitter to be called before parsing.

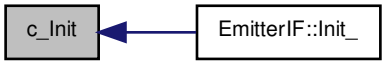
Parameters

P	the parser calling this emitter

This emitter gets called before the parser starts its parsing process. It initializes the FB source code emission.

Definition at line 764 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.21 SUB_CDECL c_exit (BYVAL_AS_Parser_PTR *P*)

Emitter to be called after parsing.

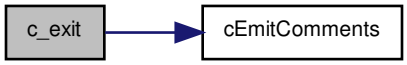
Parameters

P	the parser calling this emitter

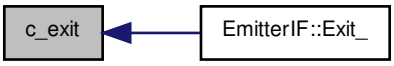
This emitter gets called after the parser ends its parsing process. It sends the rest of the FB source code to the output stream.

Definition at line 776 of file fb-doc_emitters.bas.

Here is the call graph for this function:



Here is the caller graph for this function:



15.23.2.22 SUB_CDECL c_CTOR (BYVAL_AS_Parser_PTR *P*)

CTOR to be called when starting in Options::FileModi.

Parameters

P	the parser to be used with this emitter

This CTOR gets called when starting in a mode for file input (so not for `--geany-mode`). It loads the file `fb-doc.lfn`, if any.

Definition at line 788 of file fb-doc_emitters.bas.

15.23.2.23 SUB_CDECL cEmitSource (BYVAL_AS_Parser_PTR *P*, BYVAL_AS_INTEGER *E*)

Handler for exporting FreeBasic source code.

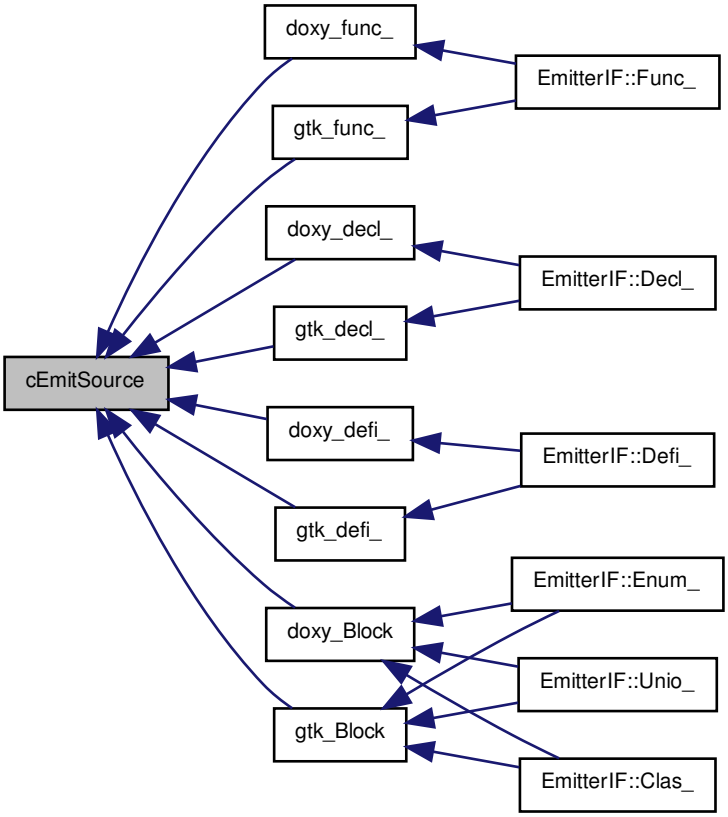
Parameters

P	the parser calling this handler
E	the end position in the input buffer

Extract original source code from the input buffer Parser::Buf. The code starts at the last position and gets extracted up to the line end before the given position. This line end gets stored as the new 'last position'.

Definition at line 828 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.24 SUB_CDECL geanyInit (BYVAL_AS_Parser_PTR *P*)

Handler for initialize the export of source code.

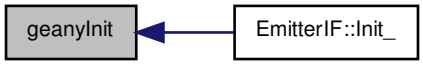
Parameters

P	the parser calling this handler

This emitter gets called before the parser starts its parsing process. It initializes the FB source code emission.

Definition at line 845 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.2.25 SUB_CDECL geanyExit (BYVAL_AS_Parser_PTR *P*)

Handler for finalize the export of source code.

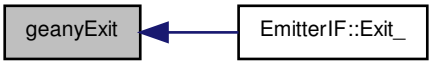
Parameters

P	the parser calling this handler

This emitter gets called after the parser ends its parsing process. It sends the rest of the FB source code to the output stream.

Definition at line 857 of file fb-doc_emitters.bas.

Here is the caller graph for this function:



15.23.3 Variable Documentation

15.23.3.1 STRING LOFN

Definition at line 25 of file fb-doc_emitters.bas.

15.24 fb-doc_emitters.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emitters.bas
00002 \brief Default emitter to create pseudo C source, \#`INCLUDE`s the other emitters
00003
00004 This file is the main file for emitters. It contains some helper
00005 functions to extract original source code and comments. And it
00006 contains the standard emitter to translate the FB source to pseudo C
00007 source.
00008
00009 The pseudo C emitter is designed to create C source out of the FB
00010 code including the documentation comments. Comments may be multi
00011 line comment blocks or line end comments. The emitter tries to place
00012 the comments similar as in the FB source (but a comment inside a
00013 statement will be placed at the end of the corresponding C statement).
00014
00015 The emitter is designed to be used to generate output for the
00016 documentation tool-chain back-end. This may be written to the STDOUT
00017 pipe (default) or to one or more files (option `--fileoutout`). It
00018 also can be used to translate an FB header to C syntax by unsing
00019 option `--cstyle`.
00020
00021 '/
00022
00023 #INCLUDE ONCE "fb-doc_emitters.bi"
00024
00025 DIM SHARED AS STRING LOFN
00026
00027 /'* \brief Handler for exporting comments
00028 \param P the parser calling this handler
00029 \param Stop_ the end position in the input buffer
00030
00031 Export the comments between the last position 'SrcBgn' (= source begin)
00032 and the Stop_ position. Then, the Stop_ position gets the new 'last
00033 position'.
00034
00035 '/
00036 SUB cEmitComments CDECL(BYVAL P AS Parser PTR, BYVAL Stop_ AS INTEGER)
00037   WITH *P
```

```
00038     VAR i = .SrcBgn
00039     WHILE i <= Stop_
00040       SELECT CASE AS CONST .Buf[i]
00041       CASE 0 : EXIT WHILE
00042       CASE ASC(!"\n") : Code(NL)
00043       CASE ASC("""")
00044         SCAN_QUOTE(.Buf,i)
00045       CASE ASC("'")
00046         VAR c = i + 1
00047         SCAN_SL_COMM(.Buf,i)
00048         IF .Buf[c] = OPT->JoComm THEN c += 1 : Code("//!" & MID(.Buf, c + 1, i - c))
00049         IF .Buf[c] = OPT->AdLine THEN c += 1 : Code(MID(.Buf, c + 1, i - c))
00050         CONTINUE WHILE
00051       CASE ASC("/") : IF .Buf[i + 1] <> ASC("'") THEN EXIT SELECT
00052         i += 2
00053         VAR c = IIF(.Buf[i] = OPT->JoComm, i + 1, 0)
00054         IF c THEN Code("/**")
00055         DO
00056           SELECT CASE AS CONST .Buf[i]
00057           CASE 0 : EXIT DO
00058           CASE ASC(!"\n") : IF 0 = c THEN Code(NL) : EXIT SELECT
00059             Code(MID(.Buf, c + 1, i - c + 1))
00060             c = i + 1
00061             IF OPT->Asterix THEN Code("* ")
00062           CASE ASC("'")
00063             SELECT CASE AS CONST .Buf[i + 1]
00064             CASE 0 : EXIT SUB
00065             CASE ASC("/") : i += 1 : EXIT DO
00066             END SELECT
00067           END SELECT : i += 1
00068         LOOP
00069         IF c THEN Code(MID(.Buf, c + 1, i - c - 1) & "*/")
00070       END SELECT : i += 1
00071     WEND : IF .SrcBgn < i THEN .SrcBgn = i
00072   END WITH
00073 END SUB
00074
00075
00076 /'* \brief Export the name (including double colons for member functions)
00077 \param P the parser calling this emitter
00078
00079 This property reads the name of a construct from the input buffer
00080 and emits all words, concatenated by double colons.
00081
00082 \returns the name including dots (colons)
00083
00084 '/
00085 SUB cppNam CDECL(BYVAL P AS Parser PTR)
00086   WITH *P
00087     VAR t = .NamTok
00088     Code(.SubStr(t))
00089     WHILE t < .EndTok
00090       t += 3
00091       SELECT CASE AS CONST *t
00092       CASE  .TOK_DOT : Code("::")
00093       CASE .TOK_WORD : Code(.SubStr(t))
00094       CASE ELSE : EXIT WHILE
00095       END SELECT
00096     WEND
00097   END WITH
00098 END SUB
00099
00100
00101 /'* \brief Export the name (including dots for member functions)
00102 \param P the parser calling this emitter
00103
00104 This sub reads the name of a construct from the input buffer
00105 and emits all words as in the original source.
00106
00107 \returns the name including dots (colons)
00108
00109 '/
00110 SUB cNam CDECL(BYVAL P AS Parser PTR)
00111   WITH *P
00112     VAR t = .NamTok, a = t[1], l = t[2]
00113     WHILE t < .EndTok
00114       t += 3
00115       SELECT CASE AS CONST *t
00116       CASE .TOK_DOT, .TOK_WORD : l += t[2]
00117       CASE ELSE : EXIT WHILE
00118       END SELECT
00119     WEND : Code(MID(.Buf, a + 1, l))
00120   END WITH
00121 END SUB
00122
00123
00124 /'* \brief Export the initializer(s) for a variable
```

```
00125 \param P the parser calling this emitter
00126
00127 This sub reads the initializer from the input buffer and emits it
00128 unchanged. This may be a bunch of text in case of an array initializer.
00129
00130 '/
00131 SUB cIni CDECL(BYVAL P AS Parser PTR)
00132   WITH *P
00133     VAR kl = 0, i = .IniTok[1], a = i, e = 0
00134     Code(" ")
00135     DO
00136       i += 1
00137       SELECT CASE AS CONST .Buf[i]
00138       CASE 0 : EXIT DO
00139       CASE ASC(!"\n")
00140         Code(MID(.Buf, a + 1, IIF(e, e, i) - a))
00141         a = i + 1
00142         IF kl <= 0 ANDALSO (e = 0 ORELSE .Buf[e] <> ASC("_")) THEN EXIT SUB
00143         cEmitComments(P, i)
00144         e = 0
00145       CASE ASC("_")
00146         IF .Buf[i + 1] < ASC("0") ANDALSO .Buf[i - 1] < ASC("0") THEN e = IIF(e, e, i)
00147       CASE ASC("'")
00148         e = IIF(e, e, i)
00149       CASE ASC("/")
00150         IF .Buf[i + 1] = ASC("'") THEN e = IIF(e, e, i)
00151       CASE ASC("""")
00152         VAR esc = IIF(.Buf[i - 1] = ASC("!"), 1, 0)
00153         DO
00154           i += 1
00155           SELECT CASE AS CONST .Buf[i]
00156           CASE 0, ASC(!"\n") : i -= 1 : EXIT DO
00157           CASE ASC("\") : IF esc THEN i += 1
00158           CASE ASC("""") : IF .Buf[i + 1] = ASC("""") THEN i += 1 ELSE EXIT DO
00159           END SELECT
00160         LOOP
00161       CASE ASC("("), ASC("{"), ASC("[") : kl += 1
00162       CASE ASC(")"), ASC("}"), ASC("]") : kl -= 1 : IF kl <  0 THEN EXIT DO
00163       CASE ASC(",")                               : IF kl <= 0 THEN EXIT DO
00164       END SELECT
00165     LOOP UNTIL i >= .EndTok[1] : IF i <= a THEN EXIT SUB
00166     Code(MID(.Buf, a + 1, IIF(e, e, i) - a))
00167   END WITH
00168 END SUB
00169
00170
00171 '* \brief The dimension of a variable
00172 \param P the parser calling this emitter
00173
00174 This property reads the dimension of a construct from the input buffer.
00175
00176 \returns the name including round or squared brackets
00177
00178 '/
00179 SUB cArrDim CDECL(BYVAL P AS Parser PTR)
00180   WITH *P
00181     VAR t = .DimTok, kl = 0
00182     DO
00183       SELECT CASE AS CONST *t
00184       CASE .TOK_COMMA ':           Code(" + 1][")
00185         if t[-3] = .TOK_TRIDO then Code("][") _
00186                               else Code(" + 1][")
00187       CASE .TOK_BROPN : kl += 1 : Code("[")
00188         IF t[3] = .TOK_COMMA ORELSE t[3] = .TOK_BRCLO THEN _
00189           Code(MID(.Buf, t[1] + 2, t[4] - t[1] - 1))
00190       CASE .TOK_BRCLO : kl -= 1 ': Code(" + 1]") : IF kl <= 0 THEN EXIT DO
00191         if t[-3] = .TOK_TRIDO then Code("]") _
00192                               else Code(" + 1]")
00193         IF kl <= 0 THEN EXIT DO
00194       CASE ELSE                  : Code(.SubStr(t))
00195       END SELECT : t += 3
00196     LOOP UNTIL t >= .EndTok
00197   END WITH
00198 END SUB
00199
00200
00201 '* \brief Create pseude C declaration
00202 \param P the parser calling this emitter
00203
00204 Create a declaration for the construct at the current parser
00205 position using FB style. All FB keywords gets mangled to a single
00206 word to get a FreeBasic look-and-feel in the documentation. Ie we emit
00207
00208  - "SUB Name();" (instead of "void Name(void);")
00209  - "INTEGER varnam" (instead of "int varname")
00210  - "BYREF_AS_STRING strng" (instead of "char** strng")
00211  - "FUNCTION_CDECL_AS_SINGLE xyz CDECL(BYVAL_AS_BYTE C)" (instead of
```

```
00212    "float xyz(char C)")
00213  - ...
00214
00215 The C lexer of the back-end (gtk-doc or Doxygen) interprets this
00216 single word as a C type declaration (or macro) and can handle FB
00217 source code that way.
00218
00219 Exeptions handled in this SUB:
00220
00221  - TypTok = 0: no type, emit only the name (with a space in from)
00222  - NymTok = 0: no name, emit type and exit
00223
00224 '/
00225 SUB cppCreateTypNam CDECL(BYVAL P AS Parser PTR)
00226   WITH *P
00227     IF .TypTok THEN
00228       IF .Co1Tok THEN Code(.SubStr(.Co1Tok) & "_")
00229                       Code(.SubStr(.TypTok))
00230       IF .Co2Tok THEN Code("_" & .SubStr(.Co2Tok))
00231       FOR i AS INTEGER = 1 TO .PtrCount
00232                       Code("_" & .SubStr(.PtrTok)) : NEXT
00233       IF .NamTok THEN Code(" ") ELSE EXIT SUB
00234     ELSE
00235       Code(" ")
00236     END IF
00237
00238     IF .NamTok > .TypTok _
00239       THEN cEmitComments(P, .NamTok[1])
00240
00241     cppNam(P)
00242     IF .DimTok THEN   cArrDim(P) 'Code(.Bracket(.DimTok))
00243     IF .BitTok THEN   Code(.BitIni)
00244     'IF .IniTok THEN   Code(.VarIni)
00245     IF .IniTok THEN   cIni(P)
00246   END WITH
00247 END SUB
00248
00249
00250 /'* \brief Create C declaration
00251 \param P the parser calling this emitter
00252
00253 Create a C declaration for the construct at the current parser
00254 position. All FB keywords gets translated to their C expressions. Ie
00255 we emit
00256
00257  - "void Name(void);" (instead of "SUB Name();")
00258  - "int varname" (instead of "INTEGER varnam")
00259  - "char** strng" (instead of "BYREF_AS_STRING strng")
00260  - "float xyz(char C)" (instead of "FUNCTION_CDECL_AS_SINGLE
00261     xyz CDECL(BYVAL_AS_BYTE C)")
00262  - ...
00263
00264 The C source code is very useful when you use a library compiled
00265 with FreeBasic in a other language like C or C++. fb-doc can
00266 auto-generate the header files (just check initializers and array
00267 dimensioning manually).
00268
00269 Exeptions handled in this SUB:
00270
00271  - TypTok = 0: no type, emit only the name (with a space in from)
00272  - NymTok = 0: no name, emit type and exit
00273
00274 '/
00275 SUB cCreateTypNam CDECL(BYVAL P AS Parser PTR)
00276   WITH *P
00277     IF .TypTok THEN
00278       IF .Co1Tok THEN             Code("const ")
00279
00280       SELECT CASE AS CONST *.TypTok
00281       CASE             .TOK_BYTE : Code("signed char")
00282       CASE             .TOK_UBYT : Code("unsigned char")
00283       CASE             .TOK_SHOR : Code("short")
00284       CASE             .TOK_USHO : Code("unsigned short")
00285       CASE             .TOK_STRI : Code("char*") ' ??? FB_STRING, g_string
00286       CASE             .TOK_SING : Code("float")
00287       CASE             .TOK_DOUB : Code("double")
00288       CASE             .TOK_ZSTR : Code("char")
00289       CASE             .TOK_WSTR : Code("wchar")
00290       CASE .TOK_INT,   .TOK_LONG : Code("int")
00291       CASE .TOK_UINT,  .TOK_ULNG : Code("unsigned int")
00292       CASE             .TOK_LINT : Code("long int")
00293       CASE             .TOK_ULIN : Code("unsigned long int")
00294       CASE ELSE                  : Code(.SubStr(.TypTok))
00295       END SELECT
00296
00297       IF .Co2Tok THEN             Code(" " & .SubStr(.Co2Tok))
00298       IF .By_Tok ANDALSO *.By_Tok = .TOK_BYRE THEN .PtrCount += 1
```

```
00299       FOR i AS INTEGER = 1 TO .PtrCount
00300                                   Code("*") : NEXT
00301       IF .NamTok THEN Code(" ") ELSE EXIT SUB
00302     END IF
00303
00304     IF .NamTok > .TypTok _
00305       THEN cEmitComments(P, .NamTok[1])
00306
00307     SELECT CASE AS CONST *.StaTok
00308     CASE .TOK_TYPE, .TOK_DIM, .TOK_RDIM, .TOK_COMM, .TOK_EXRN
00309       IF .FunTok THEN Code("(*") : cNam(P) : Code(")") : EXIT SUB
00310     END SELECT
00311     cNam(P)
00312     IF .BitTok THEN Code(.BitIni)
00313     IF .DimTok THEN cArrDim(P) ': Code()
00314     IF .IniTok THEN cIni(P)
00315     'IF .IniTok THEN Code(.VarIni)
00316   END WITH
00317 END SUB
00318
00319
00320 DECLARE SUB cppCreateFunction CDECL(BYVAL AS Parser PTR)
00321 /'* \brief Handler for a parameter declaration
00322 \param P the parser calling this handler
00323
00324 Generate a declaration for a parameter list. The declaration may be
00325 empty (), may have no name (prototype declaration) or may by an
00326 ellipsis ( ... ). Initializers get emitted "as-is".
00327
00328 We emit a space in front and a comma behind the parameter. When done
00329 the first space gets replaced by a '(' and the last comma gets
00330 replaced by a ')'.
00331
00332 '/
00333 SUB cppEntryListParameter CDECL(BYVAL P AS Parser PTR)
00334   WITH *P
00335     cEmitComments(P, .Tk1[1])
00336
00337     IF .FunTok THEN
00338       IF .By_Tok THEN Code(.SubStr(.By_Tok) & "_")
00339       IF .CalTok THEN Code("_" & .SubStr(.CalTok))
00340       IF .DivTok THEN Code("_" & .SubStr(.DivTok))
00341       cppCreateFunction(P)
00342     ELSEIF .TypTok THEN
00343       IF .By_Tok THEN Code(.SubStr(.By_Tok) & "_")
00344       Code(.SubStr(.As_Tok) & "_")
00345       cppCreateTypNam(P)
00346     ELSEIF *.NamTok = .TOK_TRIDO THEN
00347       Code("...)") : EXIT SUB
00348     END IF
00349
00350     IF *.CurTok <> .TOK_BRCLO      THEN Code(", ") : EXIT SUB
00351     cEmitComments(P, .CurTok[1]) : Code(")")
00352   END WITH
00353 END SUB
00354
00355
00356 DECLARE SUB cCreateFunction CDECL(BYVAL AS Parser PTR)
00357 /'* \brief Handler for a parameter declaration
00358 \param P the parser calling this handler
00359
00360 Generate a declaration for a parameter list. The declaration may be
00361 empty (), may have no name (prototype declaration) or may by an
00362 ellipsis ( ... ). Initializers get emitted "as-is", this means you
00363 have to check if they contain FB keywords (and translate manually, if
00364 so).
00365
00366 We emit a space in front and a comma behind the parameter. When done
00367 the first space gets replaced by a '(' and the last comma gets
00368 replaced by a ')'.
00369
00370 '/
00371 SUB cEntryListParameter CDECL(BYVAL P AS Parser PTR)
00372   WITH *P
00373     cEmitComments(P, .Tk1[1])
00374
00375     IF .FunTok THEN
00376       cCreateFunction(P)
00377     ELSEIF .TypTok THEN
00378       cCreateTypNam(P)
00379     ELSEIF .NamTok = .TOK_TRIDO THEN
00380       Code("...")
00381     ELSE
00382       Code("void")
00383     END IF
00384
00385     IF *.CurTok <> .TOK_BRCLO      THEN Code(", ") : EXIT SUB
```

```
00386     cEmitComments(P, .CurTok[1]) : Code(")")
00387   END WITH
00388 END SUB
00389
00390
00391 /'* \brief Create a function declaration
00392 \param P the parser calling this handler
00393
00394 Generate a declaration for a function (SUB, FUNCTION, DESTRUCTOR,
00395 CONSTRUCTOR, PROPERTY). We emit a type, a name and a parameter list.
00396
00397 Exceptions:
00398
00399  - in a declaration a CONSTRUCTOR or DESTRUCTOR have no name
00400
00401 '/
00402 SUB cppCreateFunction CDECL(BYVAL P AS Parser PTR)
00403   WITH *P
00404     IF .DivTok THEN Code(.SubStr(.DivTok) & " ")
00405
00406     SELECT CASE AS CONST *.FunTok
00407     CASE .TOK_CTOR
00408       IF .NamTok THEN Code(.SubStr(.NamTok) & "::" & .SubStr(.NamTok)) ELSE _
00409                       Code(.BlockNam)
00410     CASE .TOK_DTOR
00411       IF .NamTok THEN Code(.SubStr(.NamTok) & "::~" & .SubStr(.NamTok)) ELSE _
00412                       Code("~" & .BlockNam)
00413     CASE ELSE
00414                       Code(.SubStr(.FunTok))
00415       IF .CalTok THEN Code("_" & .SubStr(.CalTok))
00416       IF .AliTok THEN Code("_" & .SubStr(.AliTok))
00417       IF .TypTok ANDALSO .As_Tok THEN _
00418                       Code("_" & .SubStr(.As_Tok) & "_")
00419       cppCreateTypNam(P)
00420     END SELECT : IF 0 = .ParTok THEN Code("(void)") : EXIT SUB
00421
00422     cEmitComments(P, .ParTok[1])
00423     Code(" (")
00424     .parseListPara(@cppEntryListParameter)
00425     IF .ListCount <= 0 THEN Code(")")
00426   END WITH
00427 END SUB
00428
00429
00430 /'* \brief Create a function declaration
00431 \param P the parser calling this handler
00432
00433 Generate a declaration for a function (SUB, FUNCTION, DESTRUCTOR,
00434 CONSTRUCTOR, PROPERTY). We emit a type, a name and the parameter list.
00435
00436 Exceptions:
00437
00438  - in a declaration a CONSTRUCTOR or DESTRUCTOR has no name. We use the
00439    block name instead.
00440
00441 '/
00442 SUB cCreateFunction CDECL(BYVAL P AS Parser PTR)
00443   WITH *P
00444     IF .DivTok THEN Code(.SubStr(.DivTok) & " ")
00445
00446     SELECT CASE AS CONST *.FunTok
00447     CASE .TOK_CTOR
00448       Code(.SubStr(.FunTok) & " ")
00449       IF .NamTok THEN Code(.SubStr(.NamTok) & "." & .SubStr(.NamTok)) _
00450                  ELSE Code(.BlockNam)
00451     CASE .TOK_DTOR
00452       Code(.SubStr(.FunTok) & " ")
00453       IF .NamTok THEN Code(.SubStr(.NamTok) & "." & .SubStr(.NamTok)) _
00454                  ELSE Code(.BlockNam)
00455     CASE ELSE
00456       IF 0 = .TypTok THEN Code("void ")
00457       cCreateTypNam(P)
00458     END SELECT : IF 0 = .ParTok THEN Code("(void)") : EXIT SUB
00459
00460     cEmitComments(P, .ParTok[1])
00461     Code(" (")
00462     .parseListPara(@cEntryListParameter)
00463     IF .ListCount <= 0 THEN Code("void)")
00464   END WITH
00465 END SUB
00466
00467
00468 /'* \brief Emitter to generate an \#`INCLUDE` translation
00469 \param P the parser calling this emitter
00470
00471 This emitter gets called when the parser finds an \#`INCLUDE`
00472 statement. It creates a C translation and sends it to the output
```

```
00473 stream. When option `--tree` is given it checks if the file has
00474 been done already. If not, it creates a new #Parser and starts its
00475 scanning process.
00476
00477 '/
00478 SUB c_include CDECL(BYVAL P AS Parser PTR)
00479   STATIC AS STRING done
00480   WITH *P
00481     cEmitComments(P, .Tk1[1])
00482     VAR fnam = .SubStr(.NamTok)
00483     IF OPT->Types = OPT->C_STYLE THEN
00484       VAR i = INSTRREV(fnam, ".")
00485       Code("#include " & LEFT(fnam, i))
00486       IF LCASE(RIGHT(fnam, 4)) = ".bi""" THEN Code("h""") ELSE Code("c""")
00487     ELSE
00488       Code("#include " & fnam)
00489     END IF
00490     IF OPT->InTree THEN .Include(TRIM(fnam, """"))
00491   END WITH
00492 END SUB
00493
00494
00495 /'* \brief Emitter to generate a macro translation
00496 \param P the parser calling this emitter
00497
00498 This emitter gets called when the parser finds a macro (\#`DEFINE`
00499 \#`MACRO`). It generates a C translation of the macro and sends it
00500 to the output stream.
00501
00502 '/
00503 SUB c_defi_ CDECL(BYVAL P AS Parser PTR)
00504   WITH *P
00505     cEmitComments(P, .Tk1[1])
00506
00507     Code("#define ")
00508     VAR e = .EndTok[1]
00509     IF *.StaTok = .TOK_MACR THEN
00510       VAR a = .NamTok[1], l = .CurTok[1] - a
00511       Code(MID(.Buf, a + 1, l) & "  /* (multi line FreeBasic #MACRO) ")
00512       a += l
00513       Code(MID(.Buf, a + 1, e - a) & " */ ")
00514     ELSE
00515       VAR a = .NamTok[1], l = .DivTok[-2] + .DivTok[-1] - a
00516       Code(MID(.Buf, a + 1, l))
00517       a += l
00518       l = e - a
00519       IF l > 0 then Code(" /* " & MID(.Buf, a + 1, e - a) & " */")
00520     END IF
00521     .SrcBgn = e
00522   END WITH
00523 END SUB
00524
00525
00526 /'* \brief Emitter to generate a function translation
00527 \param P the parser calling this emitter
00528
00529 This emitter gets called when the parser finds a function (SUB /
00530 FUNCTION / PROPERTY / CONSTRUCTOR / DESTRUCTOR). It translates a
00531 function and its parameter list to C-like code and sends it to the
00532 output stream. The function body is either empty or contains pseudo
00533 calls.
00534
00535 '/
00536 SUB c_func_ CDECL(BYVAL P AS Parser PTR) ' ToDo: internal function calls for diagrams
00537   WITH *P
00538     cEmitComments(P, .Tk1[1])
00539
00540     OPT->CreateFunction(P)
00541     Code(" {")
00542
00543     'IF OPT->AllCallees THEN
00544     IF LEN(LOFN) THEN
00545       VAR t = .CurTok, e = .EndTok - 6
00546       WHILE t < e
00547         VAR c = t[1] - 1
00548         IF *t >= .TOK_ABST ANDALSO .Buf[c] <> ASC("@") THEN
00549           VAR l = t[2]
00550
00551 'ERROUT("HIER1: " & .SubStr(t))
00552
00553           ''VAR t = .NamTok, a = t[1], l = t[2]
00554           ''WHILE t < .EndTok
00555           'DO
00556             't += 3 : if t >= e then exit while
00557             'SELECT CASE AS CONST *t
00558             'CASE .TOK_DOT, .TOK_WORD : l += t[2]
00559             'CASE .TOK_BROPN : exit do
```

```
00560             'CASE ELSE : continue while
00561             'END SELECT
00562           'LOOP
00563           ''WEND
00564
00565           VAR s = UCASE(MID(.Buf, c + 2, l)), x = l
00566           FOR i AS INTEGER = LEN(LOFN) - 2 TO 0 STEP -1
00567             x = l - 1
00568             WHILE s[x] = (LOFN[i] AND &b11011111)
00569               x -= 1
00570               i -= 1
00571               IF x < 0 THEN
00572                 SELECT CASE AS CONST LOFN[i]
00573                 CASE ASC(!"\n")
00574                 CASE ASC(  ".")
00575                   x = i
00576                   DO : i -= 1 : LOOP UNTIL LOFN[i] = ASC(!"\n") ORELSE i < 1
00577                   l += x - i
00578                 CASE ELSE : EXIT WHILE
00579                 END SELECT
00580                 cEmitComments(P, c + 1)
00581                 Code(" " & MID(LOFN, i + 2, l) & "();")
00582                 EXIT FOR
00583               END IF
00584             WEND : DO : i -= 1 : LOOP UNTIL LOFN[i] = ASC(!"\n") ORELSE i < 1
00585           NEXT
00586         END IF : t += 3
00587       WEND
00588     END IF
00589
00590     cEmitComments(P, .EndTok[1] - 1)
00591     Code("};")
00592   END WITH
00593 END SUB
00594
00595
00596 /'* \brief Emitter to generate a declaration translation
00597 \param P the parser calling this emitter
00598
00599 This emitter gets called when the parser is in a declaration (VAR /
00600 DIM / CONST / COMMON / EXTERN / STATIC). It generates a C
00601 translation for each variable name and sends it (them) to the output
00602 stream. Documantation comments get emitted at the appropriate place.
00603 Each declaration get a single line, even if the original source code
00604 is a comma-separated list. (This may destroy line synchonisation, so
00605 it's better to place each declaration in a single line.)
00606
00607 '/
00608 SUB c_decl_ CDECL(BYVAL P AS Parser PTR)
00609   WITH *P
00610     cEmitComments(P, .Tk1[1])
00611
00612     IF 0 = .ListCount THEN
00613       SELECT CASE AS CONST *.StaTok
00614       CASE .TOK_CONS : Code("const ")
00615       CASE .TOK_STAT : Code("static ")
00616       CASE .TOK_COMM : Code("common ")
00617       CASE .TOK_EXRN : Code("extern ")
00618       CASE .TOK_TYPE : Code("typedef ")
00619         IF 0 = .FunTok ANDALSO .TypTok > .NamTok THEN Code("struct ")
00620       END SELECT
00621     END IF
00622
00623     IF     .FunTok THEN : OPT->CreateFunction(P)
00624     ELSEIF .TypTok THEN : OPT->CreateVariable(P)
00625     ELSE
00626       IF 0 = .ListCount THEN Code("VAR ")
00627                              Code(.SubStr(.NamTok))
00628       IF .BitTok THEN        Code(.BitIni)
00629       'IF .IniTok THEN        Code(.VarIni)
00630       IF .IniTok THEN        cIni(P)
00631     END IF
00632     IF *.CurTok <= .TOK_EOS THEN Code(";") : EXIT SUB
00633     IF .NamTok > .TypTok _
00634       THEN Code(", ") _
00635       ELSE Code("; ")
00636   END WITH
00637 END SUB
00638
00639
00640 /'* \brief Handler for an enumerator entry (inside ENUM block)
00641 \param P the parser calling this handler
00642
00643 Generate an enumerator in an enum block. Name, initializers and
00644 documentation comments are emitted. Logical operators like SHL or
00645 AND are not handled jet.
00646
```

```
00647 ''/
00648 SUB cEntryBlockENUM CDECL(BYVAL P AS Parser PTR)
00649   WITH *P
00650     cEmitComments(P, .Tk1[1])
00651
00652     IF 0 = .ListCount THEN Code(STRING(.LevelCount * 2, " "))
00653     Code(.SubStr(.NamTok))
00654     'IF .IniTok THEN Code(.VarIni)
00655     IF .IniTok THEN cIni(P)
00656     IF *.CurTok <> .TOK_END THEN Code(", ")
00657   END WITH
00658 END SUB
00659
00660
00661 DECLARE SUB c_Block CDECL(BYVAL AS Parser PTR)
00662
00663 /'* \brief Handler for a context line (TYPE / UNION block)
00664 \param P the parser calling this handler
00665
00666 Generate a line in an struct or union block. Type, name,
00667 initializers and documentation comments are emitted. Logical
00668 operators like SHL or AND are not handled jet.
00669
00670 ''/
00671 SUB cEntryBlockTypeUnion CDECL(BYVAL P AS Parser PTR)
00672   WITH *P
00673     cEmitComments(P, .Tk1[1])
00674
00675     SELECT CASE AS CONST *.Tk1
00676     CASE .TOK_PRIV : Code("private:")
00677     CASE .TOK_PROT : Code("protected:")
00678     CASE .TOK_PUBL : Code("public:")
00679     CASE .TOK_ENUM, .TOK_UNIO, .TOK_TYPE, .TOK_CLAS : c_Block(P)
00680     CASE ELSE
00681       IF 0 = .ListCount    THEN Code(STRING(.LevelCount * 2, " "))
00682       IF *.Tk1 = .TOK_DECL THEN OPT->CreateFunction(P) : Code(";") : EXIT SUB
00683       IF .FunTok THEN OPT->CreateFunction(P) _
00684                  ELSE OPT->CreateVariable(P)
00685       IF *.CurTok <= .TOK_EOS THEN Code(";") : EXIT SUB
00686       IF .NamTok  >   .TypTok THEN Code(",") ELSE Code("; ")
00687     END SELECT
00688   END WITH
00689 END SUB
00690
00691
00692 /'* \brief Emitter to generate a block translation
00693 \param P the parser calling this emitter
00694
00695 This emitter gets called when the parser finds a block (TYPE / UNION
00696 / ENUM). It generates a C translation of the block and sends it to
00697 the output stream.
00698
00699 Nested blocks get parsed recursivly.
00700
00701 ''/
00702 SUB c_Block CDECL(BYVAL P AS Parser PTR)
00703   WITH *P
00704     cEmitComments(P, .Tk1[1])
00705
00706     IF .LevelCount THEN Code(STRING(.LevelCount * 2, " "))
00707     SELECT CASE AS CONST *.Tk1
00708     CASE .TOK_TYPE, .TOK_CLAS
00709       IF OPT->Types = OPT->FB_STYLE THEN
00710         Code("class " & .BlockNam & !"{ public:")
00711         .parseBlockTyUn(@cEntryBlockTypeUnion)
00712         .BlockNam = ""
00713       ELSE
00714         IF 0 = .LevelCount ANDALSO LEN(.BlockNam) THEN Code("typedef ")
00715         Code("struct " & .BlockNam & "{")
00716         .parseBlockTyUn(@cEntryBlockTypeUnion)
00717       END IF
00718     CASE .TOK_UNIO
00719       IF 0 = .LevelCount ANDALSO LEN(.BlockNam) THEN Code("typedef ")
00720       Code("union " & .BlockNam & "{")
00721       .parseBlockTyUn(@cEntryBlockTypeUnion)
00722     CASE .TOK_ENUM
00723       IF 0 = .LevelCount ANDALSO LEN(.BlockNam) THEN Code("typedef ")
00724       Code("enum " & .BlockNam & "{")
00725       .parseBlockEnum(@cEntryBlockENUM)
00726     CASE ELSE : Code("-???-")
00727     END SELECT
00728
00729     cEmitComments(P, .Tk1[1])
00730     IF .LevelCount THEN Code(STRING(.LevelCount * 2, " "))
00731     Code("};")
00732   END WITH
00733 END SUB
```

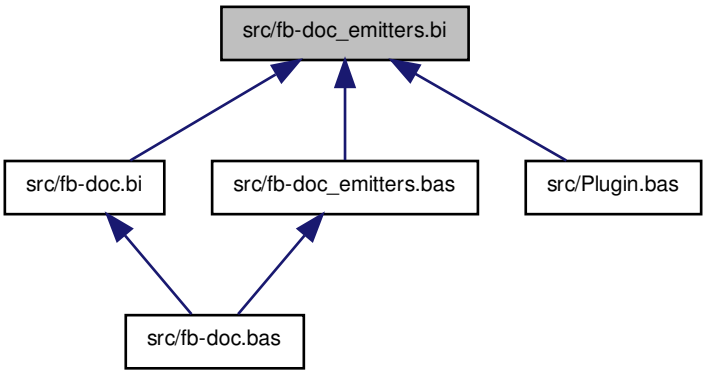
```
00734
00735
00736 /'* \brief Emitter for an error message
00737 \param P the parser calling this handler
00738
00739 Generate an error output. When the parser detects an error it calls
00740 this function. Depending on the run-mode we do or do not emit an
00741 information. In mode `--geany-mode` the error message gets shown in the
00742 status line (or in the debug window).
00743
00744 '/
00745 SUB c_error CDECL(BYVAL P AS Parser PTR)
00746   WITH *P
00747     SELECT CASE AS CONST OPT->RunMode
00748     CASE OPT->GEANY_MODE ': EXIT SUB ' or shall we output?
00749       Code(NL & "'!!! " & PROG_NAME & .ErrMsg & "!" & NL)
00750     CASE ELSE
00751       ERROUT("==> " & PROG_NAME & .ErrMsg & "!")
00752     END SELECT
00753   END WITH
00754 END SUB
00755
00756
00757 /'* \brief Emitter to be called before parsing
00758 \param P the parser calling this emitter
00759
00760 This emitter gets called before the parser starts its parsing
00761 process. It initializes the FB source code emission.
00762
00763 '/
00764 SUB c_Init CDECL(BYVAL P AS Parser PTR)
00765   P->SrcBgn = 0
00766 END SUB
00767
00768
00769 /'* \brief Emitter to be called after parsing
00770 \param P the parser calling this emitter
00771
00772 This emitter gets called after the parser ends its parsing process.
00773 It sends the rest of the FB source code to the output stream.
00774
00775 '/
00776 SUB c_exit CDECL(BYVAL P AS Parser PTR)
00777   cEmitComments(P, P->Fin)
00778 END SUB
00779
00780
00781 /'* \brief CTOR to be called when starting in \ref Options::FileModi
00782 \param P the parser to be used with this emitter
00783
00784 This CTOR gets called when starting in a mode for file input (so not
00785 for `--geany-mode`). It loads the file `fb-doc.lfn`, if any.
00786
00787 '/
00788 SUB c_CTOR CDECL(BYVAL P AS Parser PTR)
00789   VAR fnr = FREEFILE
00790   IF OPEN(CALLEES_FILE FOR INPUT AS #fnr) THEN EXIT SUB
00791   MSG_LINE(CALLEES_FILE)
00792   LOFN = STRING(LOF(fnr), 0)
00793   GET #fnr, , LOFN
00794   CLOSE #fnr
00795   MSG_END("loaded")
00796 END SUB
00797
00798
00799 ' place the handlers in the emitter interface
00800 WITH_NEW_EMITTER("C_Source")
00801
00802    .Error_ = @c_error
00803
00804     .Defi_ = @c_defi_
00805     .Incl_ = @c_include
00806     .Func_ = @c_func_
00807     .Decl_ = @c_decl_
00808     .Enum_ = @c_Block
00809     .Unio_ = @c_Block
00810     .Clas_ = @c_Block
00811
00812     .Init_ = @c_Init
00813     .Exit_ = @c_exit
00814     .CTOR_ = @c_CTOR
00815 END WITH
00816
00817
00818 /'* \brief Handler for exporting FreeBasic source code
00819 \param P the parser calling this handler
00820 \param E the end position in the input buffer
```

```
00821
00822 Extract original source code from the input buffer \ref Parser::Buf.
00823 The code starts at the last position and gets extracted up to the line
00824 end before the given position. This line end gets stored as the new
00825 'last position'.
00826
00827 '/
00828 SUB cEmitSource CDECL(BYVAL P AS Parser PTR, BYVAL E AS INTEGER)
00829   WITH *P
00830     DO : IF E > 0 THEN E -= 1 ELSE EXIT SUB
00831     LOOP UNTIL .Buf[E] = ASC(!"\n")
00832     VAR a = .SrcBgn, l = E - a : .SrcBgn = E + 1
00833     Code(MID(.Buf, a + 1, l + 1))
00834   END WITH
00835 END SUB
00836
00837
00838 /'* \brief Handler for initialize the export of source code
00839 \param P the parser calling this handler
00840
00841 This emitter gets called before the parser starts its parsing
00842 process. It initializes the FB source code emission.
00843
00844 '/
00845 SUB geanyInit CDECL(BYVAL P AS Parser PTR)
00846   P->SrcBgn = 0
00847 END SUB
00848
00849
00850 /'* \brief Handler for finalize the export of source code
00851 \param P the parser calling this handler
00852
00853 This emitter gets called after the parser ends its parsing process.
00854 It sends the rest of the FB source code to the output stream.
00855
00856 '/
00857 SUB geanyExit CDECL(BYVAL P AS Parser PTR)
00858   WITH *P
00859     Code(MID(.Buf, .SrcBgn + 1))
00860   END WITH
00861 END SUB
00862
00863
00864 ' Take care that the order of includes matches Options.EmitterTypes
00865 #INCLUDE ONCE "fb-doc_emit_callees.bas"
00866 #INCLUDE ONCE "fb-doc_emit_gtk.bas"
00867 #INCLUDE ONCE "fb-doc_emit_doxy.bas"
00868 #INCLUDE ONCE "fb-doc_emit_syntax.bas"
00869
```

15.25 src/fb-doc_emitters.bi File Reference

Header file for EmitterIF.

This graph shows which files directly or indirectly include this file:



Classes

- class EmitterIF

  The emitter interface.

Macros

- #define WITH_NEW_EMITTER(_T_)

  Snippet to create a new EmitterIF (for new customized emitter modules)

Typedefs

- typedef Parser Parser_

  Forward declaration.

- typedef SUB_CDECL EmitFunc (BYVAL_AS_Parser__PTR)

  Function to emit a piece of code.

Functions

- SUB_CDECL null_emitter (BYVAL_AS_Parser__PTR P)

  An empty emitter to initialize the interface.

Variables

- EmitterIF_PTR Emitters [TO+1]

  the SHARED array for the emitter interfaces

15.25.1 Detailed Description

Header file for EmitterIF. This file contains the declaration of the emitters interface, a UDT (User Defined Type) that contains function pointers. The Parser calls the matching function in the active emitter after scanning a relevant construct. The emitter function extracts the necessary information from the parser data, formats it as desired and sends it to the output stream.

Definition in file fb-doc_emitters.bi.

15.25.2 Macro Definition Documentation

15.25.2.1 #define WITH_NEW_EMITTER(_T_)

Value:

```
/* (multi line FreeBasic #MACRO)
 REDIM PRESERVE Emitters(UBOUND(Emitters) + 1)
 Emitters(UBOUND(Emitters)) = NEW EmitterIF
 WITH *Emitters(UBOUND(Emitters))
  .Nam = _T_
#ENDMACRO */
```

Snippet to create a new EmitterIF (for new customized emitter modules)

Definition at line 116 of file fb-doc_emitters.bi.

15.25.3 Typedef Documentation

15.25.3.1 typedef Parser Parser_

Forward declaration.

Definition at line 15 of file fb-doc_emitters.bi.

15.25.3.2 typedef SUB_CDECL EmitFunc(BYVAL_AS_Parser__PTR)

Function to emit a piece of code.

Parameters

P	the parser calling this handler

Definition at line 19 of file fb-doc_emitters.bi.

15.25.4 Function Documentation

15.25.4.1 SUB_CDECL null_emitter (BYVAL_AS_Parser__PTR *P*)

An empty emitter to initialize the interface.

Parameters

P	the parser calling this handler

Definition at line 23 of file fb-doc_emitters.bi.

15.25.5 Variable Documentation

15.25.5.1 EmitterIF_PTR Emitters[TO+1]

the SHARED array for the emitter interfaces

Definition at line 113 of file fb-doc_emitters.bi.

15.26 fb-doc_emitters.bi

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_emitters.bi
00002 \brief Header file for \ref EmitterIF
00003
00004 This file contains the declaration of the emitters interface, a UDT
00005 (User Defined Type) that contains function pointers. The \ref
00006 Parser calls the matching function in the active emitter after
00007 scanning a relevant construct. The emitter function extracts the
00008 necessary information from the parser data, formats it as desired
00009 and sends it to the output stream.
00010
00011 '/
00012
00013
00014 /'* \brief Forward declaration '/
00015 TYPE AS Parser Parser_
00016
00017 /'* \brief Function to emit a piece of code
00018 \param P the parser calling this handler '/
00019 TYPE EmitFunc AS SUB CDECL(BYVAL AS Parser_ PTR)
00020
00021 /'* \brief An empty emitter to initialize the interface
00022 \param P the parser calling this handler '/
00023 SUB null_emitter CDECL(BYVAL P AS Parser_ PTR) : END SUB
00024
00025 /'* \brief The emitter interface
00026
00027 The emitters interface is a UDT containing function pointers. The
00028 \ref Parser calls the matching function in the active emitter after
00029 scanning a relevant construct. The emitter function extracts the
00030 necessary information from the parser data, formats it as desired
00031 and sends it to the output stream.
00032
00033 Only one emitter can be active at a time. Either one of the four
00034 inbuild fb-doc emitters or an external emitter plugin can be choosen
00035 by option `--emitter`.
00036
00037 The function pointers get initialized with the null_emitter()
00038 function (SUB), that creates no output at all. Each emitter modul can
00039 replace one or more of the pointers by a customized function to
00040 create a specific output.
00041
00042 Since Doxygen doesn't support to generate documentation for such an
00043 interface, it cannot create caller or callee graphs for the emitter
00044 functions. But we use fb-doc and can work-around this by creating
00045 additional C output in form of member function. These functions are
00046 unvisible for the FreeBasic compiler, but get emitted to the pseudo C
00047 source for the Doxygen back-end and produce the desired output for
00048 the documentation.
00049
00050 '/
00051 TYPE EmitterIF
00052   AS STRING Nam = ""  '*< the emitters name
00053 ' This is tricky code to make Doxygen document an interface:
00054 '&/* Doxygen shouldn't parse this ...
00055   AS EmitFunc _
00056      Decl_ = @null_emitter _ '*< handler for `DECLARE`
00057    , Func_ = @null_emitter _ '*< handler for `FUNCTION  SUB  PROPERTY  CONSTRUCTOR  DESTRUCTOR`
00058    , Enum_ = @null_emitter _ '*< handler for `ENUM` blocks
00059    , Unio_ = @null_emitter _ '*< handler for `UNION` blocks
00060    , Clas_ = @null_emitter _ '*< handler for `TYPE  CLASS` blocks
00061    , Defi_ = @null_emitter _ '*< handler for \#`DEFINE` \#`MACRO`
00062    , Incl_ = @null_emitter _ '*< handler for \#`INCLUDE`
00063    , Init_ = @null_emitter _ '*< handler for start-up
00064   , Error_ = @null_emitter _ '*< handler for errors
00065   , Empty_ = @null_emitter _ '*< handler for empty Geany block
00066    , Exit_ = @null_emitter _ '*< handler for end-up
00067    , CTOR_ = @null_emitter _ '*< CONSTRUCTOR for emitter
00068    , DTOR_ = @null_emitter _ '*< DESTRUCTOR for emitter
00069 '& ... but the following pseudo inline members instead */
00070
00071 '* function called before parsing a source code
00072 '& inline void Init_ (void){c_Init(); geanyInit(); synt_init();};
00073
```

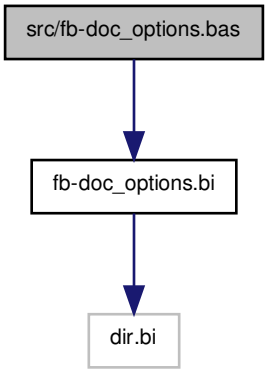
```
00074 '* emitter for a declaration (VAR / DIM / CONST / COMMON / EXTERN / STATIC)
00075 '& inline void Decl_ (void){c_decl_(); gtk_decl_(); doxy_decl_(); callees_decl_();};
00076
00077 '* emitter for a function (SUB / FUNCTION / PROPERTY / CONSTRUCTOR / DESTRUCTOR)
00078 '& inline void Func_ (void){c_func_(); gtk_func_(); doxy_func_(); callees_func_(); synt_func_();};
00079
00080 '* emitter for a ENUM block
00081 '& inline void Enum_ (void){c_Block(); gtk_Block(); doxy_Block();};
00082
00083 '* emitter for a UNION block
00084 '& inline void Unio_ (void){c_Block(); gtk_Block(); doxy_Block(); callees_class_();};
00085
00086 '* emitter for a struct (TYPE / CLASS block)
00087 '& inline void Clas_ (void){c_Block(); gtk_Block(); doxy_Block(); callees_class_();};
00088
00089 '* emitter for a macro (\#'DEFINE' / \#'MACRO')
00090 '& inline void Defi_ (void){c_defi_(); gtk_defi_(); doxy_defi_();};
00091
00092 '* emitter for includes (\#'INCLUDE')
00093 '& inline void Incl_ (void){c_include(); callees_include(); synt_incl();};
00094
00095 '* emitter for an error message
00096 '& inline void Error_ (void){c_error();};
00097
00098 '* emitter for an empty Geany block
00099 '& inline void Empty_ (void){gtk_empty(); doxy_empty(); synt_empty();};
00100
00101 '* function called after parsing the source code
00102 '& inline void Exit_ (void){c_exit(); geanyExit(); synt_exit();};
00103
00104 '* function called at program start-up (once)
00105 '& inline void CTOR_ (void){synt_CTOR();};
00106
00107 '* function called at program end (once)
00108 '& inline void DTOR_ (void){synt_DTOR();};
00109 END TYPE
00110
00111
00112 '* the SHARED array for the emitter interfaces
00113 REDIM SHARED AS EmitterIF PTR Emitters(-1 TO -1)
00114
00115 /'* \brief Snippet to create a new \ref EmitterIF (for new customized emitter modules) '/
00116 #MACRO WITH_NEW_EMITTER(_T_)
00117  REDIM PRESERVE Emitters(UBOUND(Emitters) + 1)
00118  Emitters(UBOUND(Emitters)) = NEW EmitterIF
00119  WITH *Emitters(UBOUND(Emitters))
00120   .Nam = _T_
00121 #ENDMACRO
00122
```

15.27 src/fb-doc_options.bas File Reference

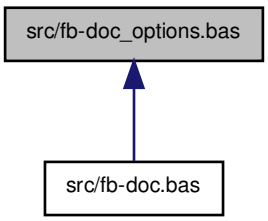
The source code for the Options class.

```
#include "fb-doc_options.bi"
```

Include dependency graph for fb-doc_options.bas:



This graph shows which files directly or indirectly include this file:



15.27.1 Detailed Description

The source code for the Options class. This file contains the source code for the Options class. It's used to scan the command line options and control the execution of fb-doc.

Definition in file fb-doc_options.bas.

15.28 fb-doc_options.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_options.bas
00002 \brief The source code for the \ref Options class
00003
00004 This file contains the source code for the Options class. It's used
00005 to scan the command line options and control the execution of fb-doc.
00006
00007 '/
00008
```

```
00009 #INCLUDE ONCE "fb-doc_options.bi"
00010
00011 /'* \brief Read options and parameters from the command line
00012
00013 The constructor scans all command line arguments and checks for
00014 options and their parameters (, see \ref sectTabOptions for
00015 details). Options may be specified in short form (starting with a
00016 single minus character) or in human readable long form (starting
00017 with two minus characters). Some options may have an additinal
00018 parameter.
00019
00020 Each command line argument that is neither an option nor its parameter
00021 gets interpreted as a file name or pattern. fb-doc collects them in
00022 a queue and operates on this queue afterwards. The queue may have
00023 mixed entries (names and patterns). It's recommended to specify
00024 queue entries in single or double quotes.
00025
00026 '/
00027 CONSTRUCTOR Options()
00028   Efnr = FREEFILE
00029   IF OPEN ERR (AS #Efnr) THEN ?PROG_NAME & ": " & "couldn't open STDERR" : EXIT CONSTRUCTOR
00030       Types = FB_STYLE
00031   CreateFunction = @cppCreateFunction
00032   CreateVariable = @cppCreateTypNam
00033 END CONSTRUCTOR
00034
00035
00036 /'* \brief The destructor
00037
00038 Delete the memory used for an external emitter (if any) and for the
00039 Parser.
00040
00041 '/
00042 DESTRUCTOR Options()
00043   IF DllEmitter THEN DYLIBFREE(DllEmitter)
00044   IF Pars THEN DELETE Pars
00045   CLOSE #Efnr
00046 END DESTRUCTOR
00047
00048
00049 /'* \brief parse an additional parameter for an option
00050 \param Idx the current index in `COMMAND()`
00051 \returns the parameter, without quotes if any
00052
00053 This function evaluates a parameter for an option. Some options need
00054 an additional parameter (ie like `--outpath`). It gets read by this
00055 function, removing surrounding single or double quotes (if any).
00056
00057 In case of no further parameter or when the parameter starts by an
00058 "`-`" character an error messages gets created.
00059
00060 '/
00061 FUNCTION Options.parseOptpara(BYREF Idx AS INTEGER) AS STRING
00062   VAR p = Idx + 1
00063   IF LEN(COMMAND(p)) ANDALSO LEFT(COMMAND(p), 1) <> "-" THEN
00064     Idx = p
00065     SELECT CASE AS CONST ASC(COMMAND(p))
00066     CASE ASC(""""), ASC("'") : RETURN MID(COMMAND(p), 2, LEN(COMMAND(p)) - 2)
00067     END SELECT : RETURN COMMAND(p)
00068   END IF : Errr = ", parameter expected [" & COMMAND(Idx) & "]" : RETURN ""
00069 END FUNCTION
00070
00071
00072 /'* \brief parse the command line
00073 \returns the RunMode
00074
00075 This function parses the command line. Options and its parameter (if
00076 required) get read and the file specifier(s) are listred in the
00077 variable \ref InFiles, separated by new line characters.
00078
00079 The function returns the value of \ref RunMode. In case of an error
00080 the RunMode is \ref ERROR_MESSAGE and the variable \ref Errr
00081 contains an message text. fb-doc stops execution in that case.
00082
00083 Loading an external emitter plugin must be done after scanning the
00084 COMMAND strings because DYLIBLOAD destroys the COMMAND array.
00085
00086 '/
00087 FUNCTION Options.parseCLI() AS RunModes
00088   VAR i = 1, emi = ""
00089   WHILE LEN(COMMAND(i)) '                                  evaluate options
00090     SELECT CASE AS CONST ASC(COMMAND(i))
00091     CASE ASC("-")
00092       SELECT CASE COMMAND(i)
00093       CASE "-f", "--file-mode"
00094         IF RunMode <> DEF_MODE THEN Errr &= ", multiple run modes" : RETURN ERROR_MESSAGE
00095         RunMode = FILE_MODE
```

```
00096         EmitTyp = C_SOURCE
00097       CASE "-g", "--geany-mode"
00098         IF RunMode <> DEF_MODE THEN Errr &= ", multiple run modes" : RETURN ERROR_MESSAGE
00099         RunMode = GEANY_MODE
00100         EmitTyp = GTK_DOC_TEMPLATES
00101         emi = parseOptpara(i)
00102       CASE "-l", "--list-mode"
00103         IF RunMode <> DEF_MODE THEN Errr &= ", multiple run modes" : RETURN ERROR_MESSAGE
00104         RunMode = LIST_MODE
00105         EmitTyp = FUNCTION_NAMES
00106       CASE "-s", "--syntax-mode"
00107         IF RunMode <> DEF_MODE THEN Errr &= ", multiple run modes" : RETURN ERROR_MESSAGE
00108         RunMode = SYNT_MODE
00109         EmitTyp = SYNTAX_REPAIR
00110
00111       CASE "-a", "--asterix" : Asterix = 1
00112       CASE "-c", "--cstyle"
00113             Types = C_STYLE
00114         CreateFunction = @cCreateFunction
00115         CreateVariable = @cCreateTypNam
00116       CASE "-t", "--tree" : InTree = 1
00117
00118       CASE "-e", "--emitter"
00119         IF LEN(emi) THEN Errr &= ", multiple emitter setting" : RETURN ERROR_MESSAGE
00120         emi = parseOptpara(i)
00121         IF 0 = LEN(emi) THEN Errr &= ", invalid emitter setting" : RETURN ERROR_MESSAGE
00122       CASE "-o", "--outpath"
00123         IF LEN(OutPath) THEN Errr &= ", multiple outpaths" : RETURN ERROR_MESSAGE
00124         OutPath = parseOptpara(i)
00125         IF 0 = LEN(OutPath) THEN Errr &= ", invalid outpath" : RETURN ERROR_MESSAGE
00126       CASE "-r", "--recursiv" : InRecursiv = 1
00127
00128       CASE "-h", "--help" : RETURN HELP_MESSAGE
00129       CASE "-v", "--version" : RETURN VERSION_MESSAGE
00130
00131       CASE ELSE
00132         ERROUT("unknown option: " & COMMAND(i))
00133       END SELECT
00134     CASE ELSE
00135       SELECT CASE AS CONST ASC(COMMAND(i))
00136       CASE ASC("""), ASC("'")
00137                   InFiles &= MID(COMMAND(i), 2, LEN(COMMAND(i)) - 2) & !"\n"
00138       CASE ELSE : InFiles &= COMMAND(i) & !"\n"
00139       END SELECT
00140     END SELECT : i += 1
00141   WEND
00142
00143   IF LEN(emi) THEN chooseEmitter(emi) ELSE EmitIF = Emitters(EmitTyp)
00144
00145   IF 0 = LEN(InFiles) ANDALSO RunMode = DEF_MODE THEN RunMode = HELP_MESSAGE
00146   Pars = NEW Parser(EmitIF)
00147   RETURN IIF(LEN(Errr), ERROR_MESSAGE, RunMode)
00148 END FUNCTION
00149
00150
00151 /'* \brief Choose a named emitter, if possible
00152 \param F the name of the emitter to search for
00153
00154 This function checks for an emitter specified by the parameter F. The
00155 check is not case-sensitve and is done for a complete emitter name
00156 as well as for a fragment. So it's enough to specify some of the
00157 start characters of the emitter name (ie *dox* instead of \em
00158 DoxygenTemplates).
00159
00160 In case of no match in any internal \ref EmitterIF::Nam this SUB tries
00161 to load an external emitter. If this fails an error message gets
00162 created and fb-doc stops execution.
00163
00164 '/
00165 SUB Options.chooseEmitter(BYREF F AS STRING)
00166   VAR t = UCASE(F), l = LEN(t)
00167   FOR e AS INTEGER = UBOUND(Emitters) TO 0 STEP -1
00168     IF t = UCASE(LEFT(Emitters(e)->Nam, l)) THEN
00169       EmitTyp = e
00170       EmitIF = Emitters(e)
00171       EXIT SUB
00172     END IF
00173   NEXT
00174
00175 #IFDEF __FB_DOS__
00176   Errr &= ", no plugin support on DOS platform!"
00177 #ELSE
00178   DllEmitter = DYLIBLOAD(f)
00179   IF DllEmitter THEN
00180     DIM ini AS FUNCTION CDECL() AS EmitterIF PTR = DYLIBSYMBOL(DllEmitter, "EMITTERINIT")
00181     IF ini THEN EmitIF = ini() : EmitTyp = EXTERNAL : EXIT SUB
00182     Errr &= ", no EMITTERINIT function in emitter " & F
```

```
00183   END IF
00184   Errr &= ", couldn't find plugin " & F
00185 #ENDIF
00186 END SUB
00187
00188
00189 /'* \brief Scan file names of given pattern
00190 \param Patt The name pattern to search for
00191 \param Path A path to add
00192 \returns A list of file names and their subfolder
00193
00194 This function scans the current directory for file names matching a
00195 given pattern and the subfolders, if \ref Options::InRecursiv ist set.
00196 It returns a list of filenames including their subfolder. The list is
00197 separated by newline characters.
00198
00199 '/
00200 FUNCTION Options.scanFiles(BYREF Patt AS STRING, BYREF Path AS STRING) AS STRING
00201   STATIC AS STRING p
00202   VAR path_l = LEN(p), f = ""
00203   IF LEN(Path) THEN p &= Path & SLASH
00204
00205   IF InRecursiv > 0 THEN
00206     VAR res = 0, n = DIR("*", fbDirectory, res), t = ""
00207     WHILE LEN(n)
00208       IF res = fbDirectory ANDALSO n <> "." ANDALSO n <> ".." THEN t &= n & NL
00209       n = DIR()
00210     WEND
00211
00212     VAR a = 1, e = a, l = LEN(t)
00213     WHILE a < l
00214       e = INSTR(a, t, NL)
00215       n = MID(t, a, e - a)
00216       IF 0 = CHDIR(n) THEN f &= scanFiles(Patt, n) : CHDIR ("..")
00217       a = e + 1
00218     WEND
00219   END IF
00220
00221   VAR n = DIR(Patt), f_l = LEN(f)
00222   WHILE LEN(n)
00223     f &= p & n & NL
00224     n = DIR()
00225   WEND
00226
00227   IF RunMode = FILE_MODE ANDALSO f_l > LEN(f) ANDALSO checkDir(OutPath & p) then f = ""
00228   p = LEFT(p, path_l)
00229   RETURN f
00230 END FUNCTION
00231
00232
00233 /'* \brief Add two directories
00234 \param P1 the path of the basic directory
00235 \param P2 the path of the directory to add
00236 \returns a string of the combined directory
00237
00238 Append second directory to first, if it's not an absolute path. In case
00239 of an absolute path this path gets returned. The returned path has a
00240 SLASH at the end and all '..' sequences are removed.
00241
00242 '/
00243 FUNCTION Options.addPath(BYREF P1 AS STRING, BYREF P2 AS STRING) AS STRING
00244   IF LEN(P1) ANDALSO RIGHT(P1, 1) <> SLASH THEN P1 &= SLASH
00245   IF 0 = LEN(P2) ORELSE P2 = "." THEN RETURN P1
00246   IF RIGHT(P2, 1) <> SLASH THEN P2 &= SLASH
00247 #IFDEF __FB_UNIX__
00248   IF P2[0] = ASC("/") THEN RETURN P2
00249 #ELSE
00250   IF MID(P2, 2, 1) = ":" THEN RETURN P2
00251 #ENDIF
00252   VAR i = LEN(P1), s = 1
00253   WHILE MID(P2, s, 3) = *DirUp
00254     i = INSTRREV(P1, SLASH, i - 1) : IF 0 = i THEN RETURN MID(P2, s)
00255     s += 3
00256   WEND
00257   IF MID(P2, s, 2) = MID(*DirUp, 2) THEN s += 2
00258   RETURN LEFT(P1, i) & MID(P2, s)
00259 END FUNCTION
00260
00261
00262 /'* \brief Create folder (if not exists)
00263 \param P the folder name
00264 \returns -1 on error, else 0
00265
00266 This FUNCTION gets called to prepare the folders for file output in
00267 `--file-mode`. It creates matching subfolders in the target
00268 directory set by option `--outpath` or its default.
00269
```

```
00270 '/
00271 FUNCTION Options.checkDir(BYREF P AS STRING) AS INTEGER
00272   VAR a = 1, e = a, l = LEN(P), cupa = CURDIR()
00273 #IFDEF __FB_UNIX__
00274   IF LEFT(P, 1) = "/" THEN a = 2 : CHDIR("/")
00275 #ELSE
00276   IF MID(P, 2, 1) = ":" THEN a = 4 : CHDIR(LEFT(P, 3))
00277 #ENDIF
00278
00279   DO
00280     e = INSTR(a, P, SLASH) : IF e = 0 THEN e = l + 1
00281     VAR n = MID(P, a, e - a)
00282     IF 0 = CHDIR(n) THEN a = e + 1 : CONTINUE DO
00283     MKDIR(n)
00284     IF 0 = CHDIR(n) THEN a = e + 1 : CONTINUE DO
00285     EXIT DO
00286   LOOP UNTIL a > l : CHDIR(cupa) : RETURN IIF(a > l, 0, 1)
00287 END FUNCTION
00288
00289
00290 /'* \brief Operate on file(s) and / or pattern(s)
00291
00292 This SUB gets called in case of file input modes. It separates the file
00293 name specifiers from the \ref Options::InFiles list, expands file
00294 patterns and executes the required operations.
00295
00296 '/
00297 SUB Options.FileModi()
00298   'Efnr = FREEFILE
00299   'IF OPEN ERR (AS #Efnr) THEN ?PROG_NAME & ": " & "couldn't open STDERR" : EXIT SUB
00300   StartPath = CURDIR()
00301   VAR cupa = StartPath
00302   EmitIF->CTOR_(Pars)
00303
00304   IF 0 = LEN(InFiles) THEN '                              InFiles defaults
00305     SELECT CASE AS CONST RunMode
00306     CASE SYNT_MODE, LIST_MODE : InFiles = !"Doxyfile\n"
00307     CASE ELSE : InFiles = !"*.bas\n*.bi\n"
00308     END SELECT
00309   END IF
00310
00311   IF 0 = LEN(OutPath) ANDALSO RunMode = FILE_MODE THEN
00312     OutPath = ".." & SLASH & "doc" & SLASH '   def. OutPath in file mode
00313     SELECT CASE AS CONST EmitTyp '          + emitter specific extension
00314     CASE C_SOURCE      : OutPath &= "c_src"
00315     CASE SYNTAX_REPAIR : OutPath &= "fb_html"
00316     CASE ELSE          : OutPath &= "src"
00317     END SELECT
00318   END IF
00319   OutPath = addPath(cupa, OutPath)
00320
00321   IF RunMode = DEF_MODE THEN
00322     Ocha = FREEFILE
00323     OPEN CONS FOR OUTPUT AS #Ocha
00324   END IF
00325
00326   VAR a = 0 _              ' Start character of next file name / pattern
00327     , i = a _              ' Counter for characters
00328     , inslsh = 0 _         ' Flag, set when filename name contains a path
00329     , inpat = 0 _          ' Flag, set when name is a pattern
00330     , l = LEN(InFiles) - 1 ' Length of input queue (number of characters)
00331   DO
00332     SELECT CASE AS CONST InFiles[i]
00333     CASE 0 : EXIT DO
00334     CASE ASC("*"), ASC("?") : inpat = 1
00335     CASE ASC(SLASH) : inslsh = i
00336     CASE ASC(!"\n")
00337       IF inpat THEN
00338         VAR in_pattern = ""
00339         IF inslsh THEN
00340           VAR path = MID(InFiles, a + 1, inslsh - a)
00341           IF CHDIR(path) THEN
00342             ERROUT("couldn't change dir to " & path)
00343           ELSE
00344             IF InTree THEN StartPath = addPath(cupa, path)
00345             in_pattern = scanFiles(MID(InFiles, inslsh + 2, i - inslsh - 1), "")
00346           END IF
00347         ELSE
00348           in_pattern = scanFiles(MID(InFiles, a + 1, i - a), "")
00349         END IF
00350
00351         VAR aa = 1, ee = aa, ll = LEN(in_pattern)
00352         WHILE aa < ll
00353           ee = INSTR(aa + 1, in_pattern, !"\n")
00354           doFile(MID(in_pattern, aa, ee - aa))
00355           aa = ee + 1
00356         WEND
```

```
00357         IF inslsh THEN CHDIR(cupa) : StartPath = cupa : inslsh = 0
00358       ELSE
00359         FileName = MID(InFiles, a + 1, i - a)
00360         doFile(FileName)
00361       END IF
00362       IF InTree THEN FileIncl = "" : Level = 0
00363
00364       inpat = 0
00365       a = i + 1
00366     END SELECT : i += 1
00367   LOOP
00368
00369   SELECT CASE AS CONST RunMode
00370   CASE LIST_MODE : IF Ocha THEN CLOSE #Ocha : MSG_LINE(CALLEES_FILE) : MSG_END("written")
00371   CASE ELSE      : IF Ocha THEN CLOSE #Ocha
00372   END SELECT
00373   EmitIF->DTOR_(Pars)
00374 END SUB
00375
00376
00377 /'* \brief Operate a single file
00378
00379 This SUB gets called to operate on a single file. Depending on the
00380 \ref RunMode it emits messages to STDERR and it opens a file for the
00381 output channel \ref Ocha.
00382
00383 '/
00384 SUB Options.doFile(BYREF Fnam AS STRING)
00385   SELECT CASE AS CONST RunMode
00386   CASE DEF_MODE  : Pars->File_(Fnam, InTree) : MSG_LINE(Fnam) : MSG_END(Pars->ErrMsg)
00387   CASE SYNT_MODE : VAR nix = NEW Highlighter(Pars) : nix->DoDoxy(Fnam) : DELETE nix
00388   CASE LIST_MODE
00389     if lcase(right(Fnam, 4)) = ".bas" orelse _
00390        lcase(right(Fnam, 3)) = ".bi" THEN
00391       if 0 = Ocha THEN
00392         MSG_LINE(OutPath & CALLEES_FILE)
00393         Ocha = writeLFN(OutPath)
00394         IF 0 = Ocha THEN MSG_END("error (couldn't write)") : EXIT SUB
00395         MSG_END("opened")
00396       end if
00397       Pars->File_(Fnam, InTree) : MSG_LINE(Fnam) : MSG_END(Pars->ErrMsg) : EXIT SUB
00398     end if
00399
00400     VAR path = addPath(StartPath, LEFT(Fnam, INSTRREV(Fnam, SLASH))) _
00401       , doxy = NEW Doxyfile(Fnam) _
00402       , recu = InRecursiv _
00403       , oldo = Ocha _
00404       , patt = ""
00405
00406     MSG_LINE(Fnam)
00407     IF 0 = doxy->Length THEN
00408       MSG_END(doxy->Errr) : delete doxy
00409       MSG_LINE("Doxyfile")
00410       if chdir(Fnam) then MSG_END("error (couldn't change directory)") : exit sub
00411       doxy = NEW Doxyfile("Doxyfile")
00412       IF 0 = doxy->Length THEN MSG_END(doxy->Errr) : delete doxy : exit sub
00413       path = curdir()
00414     END IF
00415
00416     InRecursiv = IIF(doxy->Tag("RECURSIVE") = "YES", 1, 0)
00417     var in_path = addPath(path, doxy->Tag("INPUT"))
00418     DELETE doxy
00419     if chdir(in_path) then
00420       MSG_END("error (couldn't change to " & in_path & ")")
00421     else
00422       patt = scanFiles("*.bas", "") & scanFiles("*.bi", "")
00423       IF 0 = LEN(patt) THEN
00424         MSG_END("error (nothing to do)")
00425       ELSE
00426         MSG_END("scanned") _
00427
00428         MSG_LINE(path & CALLEES_FILE)
00429         Ocha = writeLFN(path)
00430         IF 0 = Ocha THEN
00431           MSG_END("error (couldn't write)")
00432         else
00433           MSG_END("opened")
00434           VAR a = 1, e = a, l = LEN(patt)
00435           WHILE a < l
00436             e = INSTR(a + 1, patt, !"\n")
00437             Pars->File_(MID(patt, a, e - a), InTree)
00438             MSG_LINE(MID(patt, a, e - a)) : MSG_END("scanned")
00439             a = e + 1
00440           WEND
00441           close #Ocha : MSG_LINE(path & CALLEES_FILE) : MSG_END("written")
00442         end if
00443       end if
```

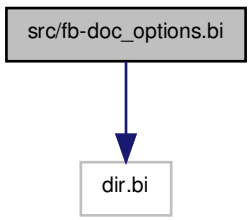
```
00444     end if
00445
00446     chdir(StartPath)
00447     Ocha = oldo
00448     InRecursiv = recu
00449
00450   CASE FILE_MODE
00451     STATIC AS STRING out_name
00452     VAR a = 1, i = INSTRREV(Fnam, ".")
00453     out_name = LEFT(Fnam, i)
00454
00455 #IFDEF __FB_UNIX__
00456     IF LEFT(out_name, 1) = "/" THEN a = 2
00457 #ELSE
00458     IF MID(out_name, 2, 1) = ":" THEN a = 2 : out_name[1] = out_name[0]
00459 #ENDIF
00460     WHILE MID(out_name, a, 3) = *DirUp
00461       a += 3
00462     WEND
00463     IF MID(out_name, a, 2) = MID(*DirUp, 2) THEN a += 2
00464     out_name = MID(out_name, a)
00465
00466     VAR path = OutPath & LEFT(out_name, INSTRREV(out_name, SLASH))
00467     IF checkDir(path) THEN
00468       ERROUT("couldn't create directory " & path)
00469     ELSE
00470       out_name = OutPath & out_name & *iif(right(FNam, 3) = ".bi", @"h", @"c")
00471       Ocha = FREEFILE
00472       IF OPEN(out_name FOR OUTPUT AS #Ocha) THEN
00473         ERROUT("couldn't write " & out_name)
00474       ELSE
00475         Pars->File_(Fnam, InTree)
00476         CLOSE #Ocha
00477         MSG_LINE(Fnam) : MSG_END(Pars->ErrMsg)
00478       END IF
00479     END IF
00480   END SELECT
00481 END SUB
00482
```

15.29 src/fb-doc_options.bi File Reference

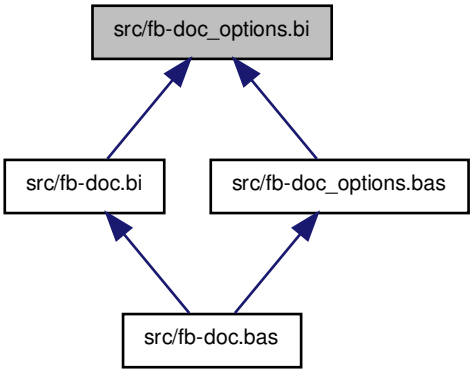
Header file for options.

`#include "dir.bi"`

Include dependency graph for fb-doc_options.bi:



This graph shows which files directly or indirectly include this file:



Classes

- class Options

  Parameters red from the command line.

Macros

- #define ERROUT(_T_) /∗ PRINT #OPT->Efnr, PROG_NAME & ": " & _T_ ∗/

  Emit an error message to STDERR.
- #define MSG_LINE(_T_) /∗ PRINT #OPT->Efnr, SPC(38 - LEN(_T_)); _T_; " --> "; ∗/

  Emit a message to STDERR.
- #define MSG_END(_T_) /∗ PRINT #OPT->Efnr, _T_ ∗/

  Emit a message to STDERR.

Variables

- const VAR CALLEES_FILE = "fb-doc.lfn"

  File name for list of function names (caller / callees graphs)
- Options_PTR OPT

  The global struct for all parameters red from the command line.

15.29.1 Detailed Description

Header file for options. This file contains the declaration of the Options class. It's used to scan options and parameters in the command line and to control execution.

Definition in file fb-doc_options.bi.

15.29.2 Macro Definition Documentation

15.29.2.1 #define ERROUT(_T_) /* PRINT #OPT->Efnr, PROG_NAME & ": " & _T_ */

Emit an error message to STDERR.

Definition at line 14 of file fb-doc_options.bi.

15.29.2.2 #define MSG_LINE(_T_) /* PRINT #OPT->Efnr, SPC(38 - LEN(_T_)); _T_; " --> "; */

Emit a message to STDERR.

Definition at line 17 of file fb-doc_options.bi.

15.29.2.3 #define MSG_END(_T_) /* PRINT #OPT->Efnr, _T_ */

Emit a message to STDERR.

Definition at line 20 of file fb-doc_options.bi.

15.29.3 Variable Documentation

15.29.3.1 const VAR CALLEES_FILE = "fb-doc.lfn"

File name for list of function names (caller / callees graphs)

Definition at line 24 of file fb-doc_options.bi.

15.29.3.2 Options_PTR OPT

The global struct for all parameters red from the command line.

Definition at line 180 of file fb-doc_options.bi.

15.30 fb-doc_options.bi

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_options.bi
00002 \brief Header file for options
00003
00004 This file contains the declaration of the \ref Options class. It's used
00005 to scan options and parameters in the command line and to control
00006 execution.
00007
00008 '/
00009
00010 #INCLUDE ONCE "dir.bi"
00011
00012
00013 /'* \brief Emit an error message to STDERR '/
00014 #DEFINE ERROUT(_T_) PRINT #OPT->Efnr, PROG_NAME & ": " & _T_
00015
00016 /'* \brief Emit a message to STDERR '/
00017 #DEFINE MSG_LINE(_T_) PRINT #OPT->Efnr, SPC(38 - LEN(_T_)); _T_; " --> ";
00018
00019 /'* \brief Emit a message to STDERR '/
00020 #DEFINE MSG_END(_T_) PRINT #OPT->Efnr, _T_
00021
00022
00023 /'* \brief File name for list of function names (caller / callees graphs) '/
00024 CONST CALLEES_FILE = "fb-doc.lfn"
00025
00026 /'* \brief Parameters red from the command line
00027
00028 The class to scan options and parameters from command line. the
00029 command line gets parsed once at the program start. Since this
```

```
00030 structure is global, the settings are available in all internal
00031 modules.
00032
00033 '/
00034 TYPE Options
00035 /'* \brief fb-doc operation modes. '/
00036   ENUM RunModes
00037
00038 /'* \brief Report an error found in the command line
00039
00040     The parser found an error in the command line and we cannot operate.
00041     Output the error message and stop. '/
00042     ERROR_MESSAGE
00043
00044 /'* \brief Print the helptext
00045
00046     Stop after printing out some help information on how to start fb-doc
00047     and how to use options at the command line. (Option `--help`). '/
00048     HELP_MESSAGE
00049
00050 /'* \brief Print version information
00051
00052     Print out the version information and stop (Option `--version`). '/
00053     VERSION_MESSAGE
00054
00055 /'* \brief Operate in Geany mode
00056
00057     Input from STDIN and output on STDOUT. Usually this is to generate
00058     templates in Geany, but it also can be used with pipes. (Option \em
00059     --geany-mode). '/
00060     GEANY_MODE
00061
00062 /'* \brief Operate as Doxygen filter
00063
00064     Read a file and generate C-source to STDOUT. (Default mode). '/
00065     DEF_MODE
00066
00067 /'* \brief Operate in scan mode
00068
00069     Read input from one or more files, write output to files. Usually
00070     this is to generate pseudo C source output, but an alternative
00071     emitter can be specified. When no file name or pattern is specified,
00072     all *.bas and *.bi files in the current folder gets parsed. (Option
00073     `--file-mode`). '/
00074     FILE_MODE
00075
00076 /'* \brief Operate in list mode
00077
00078     Read input from one or more files, write output to a single file \em
00079     fb-doc.lfn. Generate a list of callee names in this file. When no file
00080     name or pattern is specified, all <em>*.bas</em> and <em>*.bi</em>
00081     files in the current folder gets parsed. (Option `--list-mode`). '/
00082     LIST_MODE
00083
00084 /'* \brief Operate in syntax-highlighting mode
00085
00086     Read input from one or more files, write output to a several files. As
00087     input fb-doc reads files created by Doxygen, containing the source
00088     listings in the intermediate format. The file types depend on the
00089     settings in the Doxyfile. It may be <em>*.html, *.tex</em> and
00090     <em>*.xml</em>, depending on GENERATE_HTML & SOURCE_BROWSER,
00091     GENERATE_LATEX & LATEX_SOURCE_CODE and GENERATE_XML &
00092     XML_PROGRAMLISTING. '/
00093     SYNT_MODE
00094
00095   END ENUM
00096
00097 /'* \brief In-build emitters
00098
00099   By default these four emitters are available in fb-doc. The
00100   enumerators are used for default settings in the Options class. The
00101   user can choose the emitter by option `--emitter`. The parameter
00102   gets checked against the \ref EmitterIF::Nam string (or parts of it). '/
00103   ENUM EmitterTypes
00104     C_SOURCE          '*< emit pseudo C source (default and option `--file-mode`)
00105     FUNCTION_NAMES    '*< emit a list of function names (option `--list-mode`)
00106     GTK_DOC_TEMPLATES '*< emit templates for gtk-doc (option `--geany-mode gtk`)
00107     DOXYGEN_TEMPLATES '*< emit templates for Doxygen (option `--geany-mode doxy`)
00108     SYNTAX_REPAIR     '*< fix syntax highlighting of Doxygen listings (option `--syntax-mode`)
00109     EXTERNAL          '*< external emitter loaded as plugin
00110   END ENUM
00111
00112 /'* \brief The style of the types in the C source
00113
00114   A FB type can be translated to a C type or it can be shown as a
00115   pseudo type. Ie instead of the C type void the pseudo type SUB can
00116   be used. '/
```

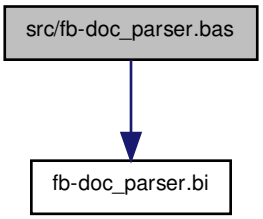
```
00117   ENUM TypesStyle
00118     C_STYLE  '*< types gets translated to C
00119     FB_STYLE '*< pseudo FB types are used
00120   END ENUM
00121
00122   '* \brief The letter-case mode for keywords
00123   ENUM CaseModes
00124     CASE_ORIGN '*< Output original formating
00125     CASE_LOWER '*< Output keywords in lower case
00126     CASE_MIXED '*< Output keywords in mixed case
00127     CASE_UPPER '*< Output keywords in upper case
00128   END ENUM
00129
00130   AS RunModes      RunMode = DEF_MODE '*< the mode to operate (defaults to DOXYFILTER)
00131   AS TypesStyle      Types = FB_STYLE '*< the style for the type generation (defaults to FB_STYLE)
00132   AS Parser PTR       Pars = 0        '*< the parser we use
00133   AS EmitterTypes  EmitTyp = C_SOURCE '*< the emitter type (defaults to \em C_Source)
00134   AS CaseModes    CaseMode = CASE_ORIGN '*< the emitter type (defaults to \em C_Source)
00135   AS EmitterIF PTR  EmitIF = 0        '*< the emitter we use (set in \ref Options::parseCLI())
00136   AS ANY PTR    DllEmitter = 0        '*< the pointer for an external emitter
00137 #IFDEF __FB_UNIX__ '& ZSTRING_PTR DirUp = "../";  /* tricky declaration for Doxygen (miss-interpretes the @ character)
00138   AS ZSTRING PTR     DirUp = @"../"   '*< sequence to get one directory up
00139 #ELSE
00140   AS ZSTRING PTR     DirUp = @"..\"   '*< sequence to get one directory up
00141 #ENDIF '&*/
00142   AS STRING _
00143        InFiles = "" _ '*< name or pattern of all input file[s]
00144     , FileName = "" _ '*< name and path of current input file
00145     , StartPath = "" _ '*< path at program start
00146     , FileIncl = "" _ '*< names of \#`INCLUDE` files
00147      , OutPath = "" _ '*< path for file output (option `--outpath`)
00148        , Errr = ""   '*< path for file output (option `--outpath`)
00149   AS INTEGER _
00150        Asterix = 0 _ '*< style for C_Source emitter to export comment blocks
00151   , AllCallees = 0 _ '*< export external callee names as well (option `--list-mode`)
00152   , InRecursiv = 0 _ '*< flag set when InFiles should get scaned recursiv in subfolders
00153      , InTree = 0 _ '*< flag set when source tree should get scaned
00154       , Level = 0 _ '*< counter for \#`INCLUDE`s
00155        , Efnr = 0 _ '*< file number for error messages (file modes)
00156        , Ocha = 0   '*< file number for output
00157   AS UBYTE _
00158     JoComm = ASC("*") _ '*< magic character to start a documentation comment
00159   , AdLine = ASC("&")   '*< magic character to start a comment for direct export
00160   AS EmitFunc _
00161     CreateFunction _  '*< emitter for a function declaration with parameter list
00162   , CreateVariable    '*< emitter for a variable declaration
00163
00164   DECLARE CONSTRUCTOR()
00165   DECLARE DESTRUCTOR()
00166   DECLARE FUNCTION parseCLI() AS RunModes
00167   DECLARE FUNCTION parseOptpara(byref Idx AS INTEGER) AS string
00168   'DECLARE SUB chooseEmitter(BYREF AS INTEGER)
00169   DECLARE SUB chooseEmitter(BYREF F AS STRING)
00170   DECLARE SUB FileModi()
00171   DECLARE SUB doFile(BYREF AS STRING)
00172   DECLARE FUNCTION checkDir(BYREF AS STRING) AS INTEGER
00173   DECLARE FUNCTION scanFiles(BYREF AS STRING, BYREF AS STRING) AS STRING
00174   DECLARE FUNCTION addPath(BYREF AS STRING, BYREF AS STRING) AS STRING
00175
00176 END TYPE
00177
00178
00179 /'* \brief The global struct for all parameters red from the command line '/
00180 DIM SHARED AS Options PTR OPT
00181
```

15.31 src/fb-doc_parser.bas File Reference

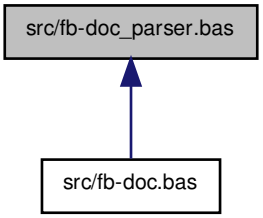
Source code for the Parser class.

```
#include "fb-doc_parser.bi"
```

Include dependency graph for fb-doc_parser.bas:



This graph shows which files directly or indirectly include this file:



Macros

- #define SKIP /∗ Tk += 3 ∗/

  Snippet to skip to the next token.
- #define SCAN_QUOTE(Buf, Po)

  Find the end of a Quote.
- #define SCAN_SL_COMM(Buf, Po)

  Find the end of a line end comment.
- #define SCAN_ML_COMM()

  Find the end of a multi line comment.
- #define SCAN_WORD()

  Find the end of a word.
- #define SETOK(_T_, _A_, _L_) /∗ Tok &= MKI(_T_) & MKI(_A_) & MKI(_L_) : ToLast = _T_ ∗/

  snippet to set the current token
- #define EXIT_STOP /∗ SETOK(MSG_STOP, Po, 1) : Po -= 1 : EXIT DO ∗/

  snippet to exit in case of end of file

15.31.1 Detailed Description

Source code for the Parser class. This file contains the source code for the Parser class. It's used to scan the FB input for relevant constructs and call the matching function in the EmitterIF.

Definition in file fb-doc_parser.bas.

15.31.2 Macro Definition Documentation

15.31.2.1 #define SKIP /* Tk += 3 */

Snippet to skip to the next token.

Definition at line 32 of file fb-doc_parser.bas.

15.31.2.2 #define SCAN_QUOTE(*Buf*, *Po*)

Value:

```
/* (multi line FreeBasic #MACRO)
  VAR esc = IIF(Buf[Po - 1] = ASC("!"), 1, 0)
  DO
    Po += 1
    SELECT CASE AS CONST Buf[Po]
    CASE 0, ASC(!"\n") : Po -= 1 : EXIT DO
    CASE ASC("\") : IF esc THEN Po += 1
    CASE ASC("""") : IF Buf[Po + 1] = ASC("""") THEN Po += 1 ELSE EXIT DO
    END SELECT
  LOOP
#ENDMACRO */
```

Find the end of a Quote.

This snippet is used to find the end of a quoted string. It checks if the string uses escape sequences and evaluates '\"' . It stops at the last double quote (if Buf[Po] isn't ASC(!"\"") then the end of the buffer is reached).

Definition at line 855 of file fb-doc_parser.bas.

15.31.2.3 #define SCAN_SL_COMM(*Buf*, *Po*)

Value:

```
/* (multi line FreeBasic #MACRO)
  DO
    Po += 1
    SELECT CASE AS CONST Buf[Po]
    CASE 0, ASC(!"\n") : EXIT DO
    END SELECT
  LOOP
#ENDMACRO */
```

Find the end of a line end comment.

This snippet is used to find the end of a line end comment. It checks for a ASC(!"\n") and evaluates line concatenations (_) on the way. It stops at the line end (if Buf[Po] isn't ASC(!"\n") then the end of the buffer is reached).

Definition at line 876 of file fb-doc_parser.bas.

15.31.2.4 #define SCAN_ML_COMM( )

Value:

```
/* (multi line FreeBasic #MACRO)
  Po += 2
```

```
  DO
    SELECT CASE AS CONST Buf[Po]
    CASE 0 : EXIT DO
    CASE ASC(!"\n") : LineNo += 1
    CASE ASC("'")
      SELECT CASE AS CONST Buf[Po + 1]
      CASE 0 : EXIT DO
      CASE ASC("/") : Po += 1 : EXIT DO
      END SELECT
    END SELECT : Po += 1
  LOOP
#ENDMACRO */
```

Find the end of a multi line comment.

This snippet is used to find the end of a multi line comment block. It stops at the characters ' and / (if Buf[Po] isn't ASC("/") then the end of the buffer is reached).

Definition at line 893 of file fb-doc_parser.bas.

15.31.2.5 #define SCAN_WORD( )

Value:

```
/* (multi line FreeBasic #MACRO)
  *A = Po
  Po += 1
  DO
    SELECT CASE AS CONST Buf[Po]
    CASE ASC("0") TO ASC("9"), _
         ASC("A") TO ASC("Z"), _
         ASC("a") TO ASC("z"), ASC("_") : Po += 1
    CASE ELSE : EXIT DO
    END SELECT
  LOOP
#ENDMACRO */
```

Find the end of a word.

This snippet is used to find the end of a word in the input buffer. Po is at the first non-word character when done.

Definition at line 915 of file fb-doc_parser.bas.

15.31.2.6 #define SETOK(*_T_*, *_A_*, *_L_*) /* Tok &= MKI(_T_) & MKI(_A_) & MKI(_L_) : ToLast = _T_ */

snippet to set the current token

Definition at line 929 of file fb-doc_parser.bas.

15.31.2.7 #define EXIT_STOP /* SETOK(MSG_STOP, Po, 1) : Po -= 1 : EXIT DO */

snippet to exit in case of end of file

Definition at line 932 of file fb-doc_parser.bas.

15.32 fb-doc_parser.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_parser.bas
00002 \brief Source code for the \ref Parser class.
00003
00004 This file contains the source code for the Parser class. It's used
00005 to scan the FB input for relevant constructs and call the matching
00006 function in the \ref EmitterIF.
00007
00008 '/
00009
00010 #INCLUDE ONCE "fb-doc_parser.bi"
```

```
00011
00012 /'* \brief The constructor
00013 \param Em a pointer to the emitter interface to use
00014
00015 Initialize the start values. We get the pointer to the \ref EmitterIF
00016 to use and we create a short token list (just two entries) for usage
00017 inside the parser. This token list is static and it doesn't change its
00018 adress (unlike the list \ref Parser::Tok that may shift when growing).
00019
00020 '/
00021 CONSTRUCTOR Parser(BYVAL Em AS EmitterIF PTR)
00022   STATIC AS INTEGER ToLi(...) = {0, 0, 0, 0, 0, 0}
00023
00024   Emit = Em
00025   StaTok = @ToLi(0)
00026   A = @ToLi(1)
00027   L = @ToLi(2)
00028 END CONSTRUCTOR
00029
00030
00031 /'* \brief Snippet to skip to the next token '/
00032 #DEFINE SKIP Tk += 3
00033
00034
00035 /'* \brief Move to the next comma and one step beyond
00036 \returns the (doubled) number of steps gone in the token list (or
00037          MSG_ERROR on error and MSG_STOP on the end of the token list)
00038
00039 We step trough the token list and search for the next comma,
00040 skipping nested pairs of parenthesis. When we find a comma we walk
00041 one step beyond. Otherwise we stop at the next right parenthesis or
00042 the next colon or line end.
00043
00044 '/
00045 FUNCTION Parser.skipOverComma() AS INTEGER
00046   VAR t = Tk, kl = 0
00047   WHILE Tk < EndTok
00048     SELECT CASE AS CONST *Tk
00049     CASE MSG_ERROR, TOK_EOS :                          RETURN Tk - t
00050     CASE TOK_COMMA : IF kl <= 0 THEN              SKIP : RETURN Tk - t
00051     CASE TOK_BROPN, TOK_KLOPN : kl += 1
00052     CASE TOK_BRCLO, TOK_KLCLO : kl -= 1 : IF kl < 0 THEN RETURN Tk - t
00053     END SELECT : SKIP
00054   WEND
00055 END FUNCTION
00056
00057
00058 /'* \brief Move to the matching right parenthesis and one step beyond
00059 \returns the (doubled) number of steps gone in the token list (or
00060          MSG_ERROR on error and MSG_STOP on the end of the token list)
00061
00062 Step through the token list and search for the next right
00063 parenthesis, skipping nested pairs of paranethesis. When we find a
00064 matching right parenthesis we walk one step beyond. Else we
00065 stop at the next colon or line end.
00066
00067 '/
00068 FUNCTION Parser.skipOverBrclo() AS INTEGER
00069   VAR t = Tk, kl = 0
00070   WHILE Tk < EndTok
00071     SELECT CASE AS CONST *Tk
00072     CASE MSG_ERROR, TOK_EOS                       : RETURN Tk - t
00073     CASE TOK_BROPN : kl += 1
00074     CASE TOK_BRCLO : kl -= 1 : IF kl <= 0 THEN SKIP : RETURN Tk - t
00075     END SELECT : SKIP
00076   WEND
00077 END FUNCTION
00078
00079
00080 /'* \brief Move to the end of statement and one step beyond
00081 \returns the (doubled) number of steps gone in the token list (or
00082          MSG_ERROR on error and MSG_STOP on the end of the token list)
00083
00084 Step through the token list and search for the end of the current
00085 statement. Then walk one step beyond, if not end of token list.
00086
00087 '/
00088 FUNCTION Parser.skipOverColon() AS INTEGER
00089   VAR t = Tk, fl = 0
00090   WHILE Tk < EndTok
00091     IF *Tk = TOK_EOS THEN fl = 1 ELSE IF fl THEN RETURN Tk - t
00092     SKIP
00093   WEND : RETURN Tk - t
00094 END FUNCTION
00095
00096
00097 /'* \brief Evaluate a name, dimension and initializer in the token list
```

```
00098 \param MinTk the token to start at
00099 \param DeclMod the modus (normal or declaration)
00100 \returns the (doubled) number of steps gone in the token list (or
00101          MSG_ERROR on error and MSG_STOP on the end of the token list)
00102
00103 Check the token list for a declaration of a name, starting at the
00104 current token (ie behind a VAR keyword). Also checking for
00105 parenthesis to specify a dimension and the equal '=' character of an
00106 initializer. After execution the current position is at the next
00107 token after the equal character, or behind the closing parenthesis,
00108 or the name. An error is returned if there is no word at the current
00109 position.
00110
00111 '/
00112 FUNCTION Parser.demuxNam(BYVAL MinTk AS INTEGER = TOK_WORD, BYVAL DeclMod AS INTEGER = 0) AS INTEGER
00113   DimTok = 0
00114   IniTok = 0
00115   BitTok = 0
00116
00117   IF *Tk >= MinTk THEN NamTok = Tk           ELSE NamTok = 0 : RETURN MSG_ERROR
00118   DO
00119     SELECT CASE AS CONST *Tk
00120     CASE TOK_BROPN, TOK_EQUAL
00121                   IF Tk > NamTok THEN EXIT DO ELSE NamTok = 0 : RETURN MSG_ERROR
00122     CASE  TOK_DOT : IF Tk[3] < MinTk         THEN NamTok = 0 : RETURN MSG_ERROR
00123     CASE TOK_WORD : IF Tk[3] <> TOK_DOT THEN SKIP : EXIT DO
00124     CASE ELSE
00125       IF 0 = LevelCount ORELSE *Tk < MinTk   THEN NamTok = 0 : RETURN MSG_ERROR
00126       SKIP : EXIT DO
00127     END SELECT : SKIP
00128   LOOP : IF DeclMod THEN                                      RETURN Tk - NamTok
00129
00130   IF *Tk = TOK_BROPN THEN DimTok = Tk : skipOverBrclo()
00131   IF *Tk = TOK_EQUAL THEN IniTok = Tk                       : RETURN Tk - NamTok
00132   IF 0 = LevelCount ORELSE _
00133      *Tk <> TOK_EOS ORELSE _
00134      Buf[Tk[1]] <> ASC(":") THEN                              RETURN Tk - NamTok
00135
00136   VAR x = Tk + 3 '                        check for bitfield declaration
00137   IF *x >= MinTk THEN
00138     x += 3
00139     IF *x = TOK_EOS ANDALSO Buf[x[1]] <> ASC(":") THEN _
00140                                       BitTok = Tk : Tk = x : RETURN Tk - NamTok
00141   END IF
00142
00143   SELECT CASE AS CONST *x
00144   CASE TOK_EQUAL : BitTok = Tk : Tk = x : IniTok = x
00145   CASE TOK_COMMA : BitTok = Tk : Tk = x
00146   CASE ELSE
00147     FOR i AS INTEGER = Tk[1] + 1 TO Tk[4] - 1
00148       SELECT CASE AS CONST Buf[i]
00149       CASE ASC(" "), ASC(!"\t"), ASC(!"\v")
00150       CASE ASC("0") TO ASC("9") : BitTok = Tk : EXIT FOR
00151       CASE ELSE : EXIT FOR
00152       END SELECT
00153     NEXT
00154   END SELECT : RETURN Tk - NamTok
00155 END FUNCTION
00156
00157
00158 /'* \brief Evaluate a function declaration in the token list
00159 \returns the the result of demuxTyp() (or MSG_ERROR on error and
00160          MSG_STOP on the end of the token list)
00161
00162 Pre-check the token list for a declaration of a function, starting at
00163 the token behind the DECLARE statement. When OK, use demuxTyp() to
00164 evaluate the type of the declaration.
00165
00166 '/
00167 FUNCTION Parser.demuxDecl() AS INTEGER
00168   SELECT CASE AS CONST *Tk
00169   CASE TOK_CONS, TOK_STAT, TOK_VIRT, TOK_ABST : DivTok = Tk : SKIP
00170   CASE ELSE : DivTok = 0
00171   END SELECT
00172
00173   FunTok = Tk : DimTok = 0 : IniTok = 0
00174   SELECT CASE AS CONST *Tk
00175   CASE TOK_CTOR, TOK_DTOR
00176     SKIP : IF *Tk = TOK_WORD THEN NamTok = Tk : SKIP ELSE NamTok = 0
00177     IF *Tk = TOK_EOS THEN RETURN TOK_EOS ELSE RETURN demuxTyp(1)
00178   CASE TOK_OPER
00179     DO
00180       SKIP
00181       IF *Tk = TOK_BROPN THEN EXIT DO
00182       IF *Tk < TOK_BROPN THEN EXIT SELECT
00183     LOOP UNTIL Tk >= EndTok : RETURN demuxTyp(1)
00184   CASE TOK_PROP
```

```
00185     SKIP : IF *Tk = TOK_WORD THEN NamTok = Tk ELSE EXIT SELECT
00186     SKIP : IF *Tk = TOK_EOS THEN RETURN TOK_EOS ELSE RETURN demuxTyp(1)
00187   CASE TOK_SUB, TOK_FUNC
00188     SKIP : IF *Tk = TOK_WORD THEN NamTok = Tk ELSE EXIT SELECT
00189     SKIP : IF *Tk = TOK_LIB THEN SKIP : IF *Tk = TOK_QUOTE THEN SKIP ELSE EXIT SELECT
00190     IF *Tk = TOK_EOS THEN RETURN TOK_EOS ELSE RETURN demuxTyp(1)
00191   END SELECT : TypTok = 0 : FunTok = 0 : NamTok = 0 : RETURN MSG_ERROR
00192 END FUNCTION
00193
00194
00195 /'* \brief Evaluate a type declaration in the token list
00196 \param DeclMod the modus (normal or declaration)
00197 \returns the (doubled) number of steps gone in the token list (or
00198          MSG_ERROR on error and MSG_STOP on the end of the token list)
00199
00200 Check the token list for a type declaration. Starting at the AS
00201 statement we read the name of the type and all following 'PTR'
00202 statements. The type may be a SUB, FUNCTION, PROPERTY, ... After
00203 execution the current position is at the next token behind the last
00204 PTR, or at the type name, or at the closing parenthesis. An error is
00205 returned if there is no word at the current position, or the token
00206 list doesn't start with 'AS', or the next token isn't a word.
00207
00208 '/
00209 FUNCTION Parser.demuxTyp(BYVAL DeclMod AS INTEGER = 0) AS INTEGER
00210   VAR t = Tk
00211   ParTok = 0
00212   CalTok = 0
00213   AliTok = 0
00214   TypTok = 0
00215   As_Tok = 0
00216   Co1Tok = 0
00217   Co2Tok = 0
00218   PtrTok = 0
00219   PtrCount = 0
00220
00221   IF 0 = DeclMod THEN
00222     DivTok = 0
00223     IF *Tk = TOK_AS THEN As_Tok = Tk : SKIP    ELSE RETURN MSG_ERROR
00224
00225     SELECT CASE AS CONST *Tk
00226     CASE TOK_SUB  : FunTok = Tk
00227     CASE TOK_FUNC : FunTok = Tk
00228     CASE ELSE
00229       IF *Tk = TOK_CONS THEN Co1Tok = Tk : SKIP
00230       IF *Tk < TOK_BYTE ORELSE *Tk > TOK_WORD THEN  RETURN MSG_ERROR
00231       TypTok = Tk : FunTok = 0
00232     END SELECT : SKIP
00233   END IF
00234
00235   SELECT CASE AS CONST *Tk
00236   CASE TOK_EQUAL
00237     IF FunTok THEN TypTok = 0                    : RETURN MSG_ERROR
00238     IniTok = Tk                                  : RETURN Tk - t
00239   CASE TOK_BROPN : IF 0 = FunTok               THEN RETURN Tk - t
00240     ParTok = Tk : skipOverBrclo()
00241   CASE ELSE
00242     IF 0 = FunTok THEN
00243       IF *Tk = TOK_CONS THEN
00244         Co2Tok = Tk : SKIP
00245         IF *Tk <> TOK_PTR THEN TypTok = 0         : RETURN MSG_ERROR
00246       ELSE
00247         IF *Tk <> TOK_PTR THEN                      RETURN Tk - t
00248       END IF
00249
00250       PtrTok = Tk : WHILE *Tk = TOK_PTR : PtrCount += 1 : SKIP : WEND
00251
00252       IF *Tk = TOK_EQUAL THEN IniTok = Tk
00253                                                     RETURN Tk - t
00254     END IF
00255
00256     IF *Tk = TOK_EOS THEN RETURN IIF(*FunTok = TOK_FUNC, MSG_ERROR, TOK_EOS)
00257
00258     SELECT CASE AS CONST *Tk
00259     CASE TOK_CDEC : CalTok = Tk : SKIP
00260     CASE TOK_STCL : CalTok = Tk : SKIP
00261     CASE TOK_PASC : CalTok = Tk : SKIP
00262     END SELECT
00263
00264     IF *Tk = TOK_OVER THEN SKIP
00265     IF *Tk = TOK_ALIA THEN
00266       AliTok = Tk
00267       SKIP : IF *Tk <> TOK_QUOTE THEN TypTok = 0  : RETURN MSG_ERROR
00268       SKIP
00269     END IF
00270
00271     IF *Tk = TOK_BROPN THEN ParTok = Tk : skipOverBrclo()
```

```
00272   END SELECT
00273
00274   IF *Tk = TOK_AS THEN
00275     As_Tok = Tk : SKIP
00276     IF *Tk = TOK_CONS THEN Co1Tok = Tk : SKIP
00277     TypTok = Tk : SKIP
00278     IF *Tk = TOK_CONS THEN
00279       Co2Tok = Tk : SKIP
00280       IF *Tk <> TOK_PTR THEN FunTok = 0: TypTok = 0 : RETURN MSG_ERROR
00281     END IF
00282     PtrTok = Tk : WHILE *Tk = TOK_PTR : PtrCount += 1 : SKIP : WEND
00283   ELSE
00284     IF *FunTok = TOK_FUNC THEN RETURN MSG_ERROR
00285   END IF ' !!! ToDo STATIC, EXPORT
00286
00287   IF *Tk = TOK_STAT THEN
00288     SKIP : IF *Tk = TOK_EXPO THEN SKIP
00289   ELSE
00290     IF *Tk = TOK_EXPO THEN SKIP : IF *Tk = TOK_STAT THEN SKIP
00291   END IF : RETURN Tk - t
00292 END FUNCTION
00293
00294
00295 /'* \brief Evaluate a list of names
00296 \param Export_ the function to call on each find
00297
00298 Scan the token list for variable declarations and call the emitter
00299 for each find  (ie for constructs like DIM AS BYTE Nam1, Nam2(5) =
00300 {0,1,2,3,4,5}, Nam3, ...).
00301
00302 '/
00303 SUB Parser.parseListNam(BYVAL Export_ AS EmitFunc) EXPORT
00304   VAR count = 1
00305   TypTok = 0 : FunTok = 0
00306   DO
00307     Tk1 = Tk
00308     IF MSG_ERROR >= demuxNam() THEN : Errr("name expected") : skipOverComma() : ELSE
00309     skipOverComma() : ListCount = count : Export_(@THIS) : count += 1 : END IF
00310   LOOP UNTIL *Tk <= TOK_COMMA
00311 END SUB
00312
00313
00314 /'* \brief Evaluate a list of declarations
00315 \param Export_ the function to call on each find
00316
00317 Scan the token list for variable declarations and call the emitter
00318 for each find  (ie for constructs like DIM Nam1 AS BYTE, Nam2(5) AS
00319 UBYTE = {0,1,2,3,4,5}, Nam3 AS STRING, ...).
00320
00321 '/
00322 SUB Parser.parseListNamTyp(BYVAL Export_ AS EmitFunc) EXPORT
00323   DO
00324     Tk1 = Tk
00325     IF MSG_ERROR >= demuxNam() THEN : Errr("name expected") : skipOverComma() : ELSE
00326     IF MSG_ERROR >= demuxTyp() THEN : Errr("type expected") : skipOverComma() : ELSE
00327     skipOverComma() : ListCount = 0 : Export_(@THIS) : END IF : END IF
00328   LOOP UNTIL *Tk <= TOK_COMMA
00329 END SUB
00330
00331
00332 /'* \brief Evaluate the context of an ENUM block
00333 \param Export_ the function to call on each find
00334
00335 The emitter calls us to evaluate statements inside an ENUM block. So
00336 we stop after finding a statement and call the emitter handler
00337 one-by-one.
00338
00339 '/
00340 SUB Parser.parseBlockEnum(BYVAL Export_ AS EmitFunc) EXPORT
00341   VAR lico = 0
00342   TypTok = 0 : FunTok = 0 : LevelCount += 1
00343   WHILE Tk < EndTok
00344     Tk1 = Tk
00345     SELECT CASE AS CONST *Tk
00346     CASE TOK_END
00347       SKIP : IF *Tk = TOK_ENUM THEN EXIT WHILE
00348       Errr("END ENUM expected") : EXIT WHILE
00349     CASE TOK_LATTE : skipOverColon()
00350     CASE ELSE
00351       IF MSG_ERROR >= demuxNam() THEN Errr("name expected") : skipOverColon() : EXIT SELECT
00352       ListCount = lico
00353       skipOverComma() : IF *Tk = TOK_EOS THEN lico = 0 : skipOverColon() ELSE lico += 1
00354       Export_(@THIS)
00355     END SELECT
00356   WEND : skipOverColon() : LevelCount -= 1
00357 END SUB
00358
```

```
00359
00360 /'* \brief Evaluate the context of a block
00361 \param Export_ the function to call on each find
00362
00363 The emitter calls us to evaluate constructs inside a block (TYPE /
00364 UNION). So we stop after finding a statement and call the emitter
00365 handler one-by-one.
00366
00367 '/
00368 SUB Parser.parseBlockTyUn(BYVAL Export_ AS EmitFunc) EXPORT
00369   VAR in_tk1 = *Tk1
00370   LevelCount += 1
00371
00372   DO
00373     Tk1 = Tk
00374     VAR nextok = Tk[3]
00375     SELECT CASE AS CONST nextok
00376     CASE TOK_AS, TOK_BROPN
00377       BitTok = 0
00378       IF MSG_ERROR >= demuxNam(TOK_ABST) THEN _
00379         IF Errr("name expected") = MSG_ERROR THEN CONTINUE DO _
00380                                              ELSE EXIT DO
00381       IF MSG_ERROR >= demuxTyp() THEN _
00382         IF Errr("type expected") = MSG_ERROR THEN CONTINUE DO _
00383                                              ELSE EXIT DO
00384       *NamTok = TOK_WORD : ListCount = 0 : Export_(@THIS)
00385       skipOverColon()
00386
00387     CASE ELSE
00388       SELECT CASE AS CONST *Tk1
00389       CASE TOK_LATTE : skipOverColon()
00390       CASE TOK_AS
00391         BitTok = 0
00392         IF MSG_ERROR >= demuxTyp() THEN _
00393           IF Errr("type expected") = MSG_ERROR THEN CONTINUE DO _
00394                                                ELSE EXIT DO
00395         IF MSG_ERROR >= demuxNam(TOK_ABST) THEN _
00396           IF Errr("name expected") = MSG_ERROR THEN CONTINUE DO _
00397                                                ELSE EXIT DO
00398
00399         skipOverComma()
00400         *NamTok = TOK_WORD : ListCount = 0 : Export_(@THIS)
00401         IF *Tk >= TOK_ABST THEN parseListNam(Export_)
00402         skipOverColon()
00403       CASE TOK_END
00404         IF nextok = in_tk1                   THEN EXIT DO
00405         IF Errr("not supported") < MSG_ERROR THEN EXIT DO
00406         skipOverColon()
00407
00408       CASE TOK_DECL : SKIP
00409         IF MSG_ERROR >= demuxDecl() THEN IF Errr("syntax error " & SubStr(NamTok)) = MSG_ERROR _
00410                                     THEN skipOverColon() : EXIT SELECT ELSE EXIT DO
00411         skipOverColon() : Export_(@THIS)
00412
00413       CASE TOK_PUBL, TOK_PRIV, TOK_PROT
00414         skipOverColon() : Export_(@THIS)
00415
00416       CASE TOK_ENUM, TOK_TYPE, TOK_UNIO, TOK_CLAS
00417         VAR n = BlockNam
00418         IF nextok = TOK_WORD THEN SKIP : BlockNam = SubStr ELSE BlockNam = ""
00419         skipOverColon() : Export_(@THIS) : BlockNam = n
00420       CASE ELSE
00421         skipOverColon() : IF Errr("not supported") < MSG_ERROR THEN EXIT DO
00422       END SELECT
00423     END SELECT
00424   LOOP UNTIL Tk >= EndTok : skipOverColon() : LevelCount -= 1
00425 END SUB
00426
00427
00428 /'* \brief Evaluate a parameter list
00429 \param Export_ the function to call on each find
00430
00431 Scan the token list for a parameter list. Evaluate declaration
00432 specifiers (BYVAL / BYREF) and handle ellipsis, parameters without
00433 name, initializers and empty lists.
00434
00435 '/
00436 SUB Parser.parseListPara(BYVAL Export_ AS EmitFunc) EXPORT
00437   VAR count = 0, t = Tk
00438   Tk = ParTok
00439   SKIP
00440   DO
00441     Tk1 = Tk
00442     By_Tok = 0
00443     SELECT CASE AS CONST *Tk
00444     CASE TOK_BRCLO : IF count > 0 THEN Errr("parameter expected")
00445       ListCount = count : EXIT DO
```

```
00446     CASE TOK_TRIDO : NamTok = Tk : TypTok = 0 : FunTok = 0
00447     CASE ELSE
00448       IF *Tk = TOK_BYVA ORELSE *Tk = TOK_BYRE THEN By_Tok = Tk : SKIP
00449       IF *Tk <> TOK_AS ANDALSO *Tk >= TOK_ABST THEN
00450         NamTok = Tk : SKIP
00451         IF *Tk = TOK_BROPN THEN DimTok = Tk : skipOverBrclo()
00452       ELSE
00453         NamTok = 0
00454       END IF
00455       IF MSG_ERROR >= demuxTyp()  THEN Errr("type expected") : skipOverComma() : CONTINUE DO
00456     END SELECT
00457     skipOverComma() : count += 1 : ListCount = count : Export_(@THIS)
00458   LOOP UNTIL *Tk < TOK_COMMA : By_Tok = 0 : Tk = t
00459 END SUB
00460
00461
00462 /'* \brief Evaluate a TYPE or CLASS statement
00463 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00464
00465 The pre-parser found a TYPE or CLASS keyword. In case of TYPE we
00466 tokenize the line and check if it is an alias declaration. In that
00467 case we call the emitter for a declaration on that single line.
00468 Otherwise we tokenize the complete block and call the emitter for a
00469 TYPE block. Or we call Errr() on syntax problems.
00470
00471 Note: the C emitter creates
00472  - 'typedef Typ Nam' for 'TYPE AS Typ Nam'
00473  - 'typedef struct Typ Nam' for 'TYPE Nam AS Typ'
00474
00475 '/
00476 FUNCTION Parser.TYPE_() AS INTEGER
00477   IF 3 > tokenize(TO_COLON) THEN RETURN Errr("syntax error")
00478
00479   DimTok = 0 : IniTok = 0
00480   IF *StaTok = TOK_TYPE THEN
00481     IF *Tk = TOK_AS THEN
00482       IF MSG_ERROR >= demuxTyp() THEN          RETURN Errr("type expected")
00483       IF *Tk = TOK_WORD THEN NamTok = Tk  ELSE RETURN Errr("name expected1")
00484       skipOverComma()
00485       Emit->Decl_(@THIS)                     : RETURN MSG_ERROR
00486     ELSE
00487       IF *Tk = TOK_WORD THEN NamTok = Tk : SKIP ELSE RETURN Errr("name expected2")
00488       IF *Tk = TOK_AS THEN
00489         IF MSG_ERROR >= demuxTyp() THEN        RETURN Errr("type expected")
00490         skipOverComma()
00491         Emit->Decl_(@THIS)                     : RETURN MSG_ERROR
00492       END IF
00493     END IF
00494   ELSE
00495     IF *Tk = TOK_WORD THEN NamTok = Tk    ELSE RETURN Errr("name expected3")
00496   END IF
00497   BlockNam = SubStr(NamTok)
00498   IF 9 > tokenize(TO_END_BLOCK) THEN           RETURN Errr("syntax error")
00499   Tk1 = StaTok
00500   Emit->Clas_(@THIS) :                         RETURN MSG_ERROR
00501 END FUNCTION
00502
00503
00504 /'* \brief Evaluate a variable declaration
00505 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00506
00507 The pre-parser found a VAR, DIM, CONST, EXTERN, COMMON or STATIC
00508 keyword. We tokenize the current line and send the result to the
00509 emitter, or we call Errr() on syntax problems.
00510
00511 '/
00512 FUNCTION Parser.VAR_() AS INTEGER
00513   IF 3 > tokenize(TO_COLON) THEN    RETURN Errr("syntax error")
00514   IF *Tk <> TOK_SHAR THEN ShaTok = 0 ELSE ShaTok = Tk : SKIP
00515
00516   IF *Tk = TOK_AS THEN
00517     IF MSG_ERROR >= demuxTyp() THEN RETURN Errr("type expected")
00518     IF MSG_ERROR >= demuxNam() THEN RETURN Errr("name expected")
00519   ELSE
00520     IF MSG_ERROR >= demuxNam() THEN RETURN Errr("name expected")
00521     IF *Tk = TOK_AS THEN demuxTyp() ELSE TypTok = 0 : FunTok = 0
00522     IF 0 = TypTok ANDALSO 0 = FunTok THEN
00523       SELECT CASE AS CONST *StaTok
00524       CASE TOK_VAR, TOK_CONS
00525         IF  0 = IniTok THEN         RETURN Errr("initialization expected")
00526       CASE ELSE                   : RETURN Errr("type expected")
00527       END SELECT
00528     END IF
00529   END IF
00530
00531   skipOverComma()
00532   Emit->Decl_(@THIS)
```

```
00533   IF *Tk <= TOK_EOS THEN RETURN MSG_ERROR
00534   IF NamTok > TypTok THEN parseListNam(Emit->Decl_) _
00535                      ELSE parseListNamTyp(Emit->Decl_)
00536   RETURN MSG_ERROR
00537 END FUNCTION
00538
00539
00540 /'* \brief Evaluate an ENUM block
00541 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00542
00543 The pre-parser found an ENUM keyword. We tokenize the context of the
00544 complete block and call the emitter, or we call Errr() on
00545 syntax problems.
00546
00547 '/
00548 FUNCTION Parser.ENUM_() AS INTEGER
00549   IF 9 > tokenize(TO_END_BLOCK) THEN RETURN Errr("syntax error")
00550   IF *Tk = TOK_WORD THEN BlockNam = SubStr : SKIP ELSE BlockNam = ""
00551   IF *Tk = TOK_EOS THEN skipOverColon() ELSE RETURN Errr("syntax errorrrr")
00552   Tk1 = StaTok
00553   Emit->Enum_(@THIS)
00554   RETURN MSG_ERROR
00555 END FUNCTION
00556
00557
00558 /'* \brief Evaluate an UNION block
00559 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00560
00561 The pre-parser found an UNION keyword. We tokenize the context of the
00562 complete block and call the emitter, or we call Errr() on
00563 syntax problems.
00564
00565 '/
00566 FUNCTION Parser.UNION_() AS INTEGER
00567   IF 9 > tokenize(TO_END_BLOCK) THEN RETURN Errr("syntax error")
00568   IF *Tk <> TOK_WORD THEN RETURN Errr("name expected")
00569   BlockNam = SubStr
00570   skipOverColon()
00571   Tk1 = StaTok
00572   Emit->Unio_(@THIS)
00573   RETURN MSG_ERROR
00574 END FUNCTION
00575
00576
00577 /'* \brief Evaluate a function
00578 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00579
00580 The pre-parser found a SUB, FUNCTION, CONSTRUCTOR, DESTRUCTOR,
00581 PROPERTY or OPERATOR keyword. We tokenize the complete block,
00582 evaluate the name and type and send the result to the emitter, or we
00583 call Errr() on syntax problems.
00584
00585 '/
00586 FUNCTION Parser.FUNCTION_() AS INTEGER
00587   SELECT CASE AS CONST ToLast
00588   CASE TOK_PUBL, TOK_PRIV : DivTok = StaTok
00589   CASE ELSE : DivTok = 0
00590   END SELECT
00591   IF 9 > tokenize(TO_END_BLOCK) THEN RETURN Errr("syntax error")
00592   FunTok = StaTok
00593
00594   IF DivTok THEN DivTok = Tk1
00595   IF MSG_ERROR >= demuxNam(TOK_WORD, 1) THEN RETURN Errr("name expected")
00596   IF MSG_ERROR >= demuxTyp(1) THEN RETURN Errr("syntax error")
00597
00598   FOR i AS INTEGER = 0 TO 1
00599     SELECT CASE AS CONST *Tk
00600     CASE TOK_STAT : SKIP ' !!! ToDo
00601     CASE TOK_EXPO : SKIP ' !!! ToDo
00602     CASE ELSE : EXIT FOR
00603     END SELECT
00604   NEXT
00605   skipOverBrclo()
00606   Emit->Func_(@THIS)
00607   RETURN MSG_ERROR
00608 END FUNCTION
00609
00610
00611 /'* \brief Evaluate a forward declaration
00612 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00613
00614 The pre-parser found a DECLARE keyword. We tokenize the current line
00615 and send the result to the emitter, or we call Errr() on syntax
00616 problems.
00617
00618 '/
00619 FUNCTION Parser.DECLARE_() AS INTEGER
```

```
00620   IF 3 > tokenize(TO_COLON) THEN RETURN Errr("syntax error")
00621   IF MSG_ERROR >= demuxDecl() THEN Errr("syntax error!!")
00622   Emit->Decl_(@THIS)
00623   RETURN MSG_ERROR
00624 END FUNCTION
00625
00626
00627 /'* \brief Evaluate an \#`INCLUDE` line
00628 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00629
00630 The pre-parser found an \#`INCLUDE` line. We tokenize the line and call
00631 the emitter, or we call Errr() handler on syntax problems. It's up
00632 to the emitter function to follow the source tree (= create a new
00633 parser and load the file).
00634
00635 '/
00636 FUNCTION Parser.INCLUDE_() AS INTEGER
00637   IF 3 > tokenize(TO_EOL) THEN RETURN Errr("syntax error")
00638   IF *Tk = TOK_ONCE THEN DivTok = Tk : SKIP ELSE DivTok = 0
00639   IF *Tk <> TOK_QUOTE THEN RETURN Errr("file name expected")
00640   NamTok = Tk
00641   Emit->Incl_(@THIS)
00642 END FUNCTION
00643
00644
00645 /'* \brief Evaluate a \#`MACRO` declaration
00646 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00647
00648 The pre-parser found a \#`MACRO` block. We tokenize the block and call
00649 the emitter, or we call Errr() handler on syntax problems.
00650
00651 '/
00652 FUNCTION Parser.MACRO_() AS INTEGER
00653   IF 3 > tokenize(TO_END_BLOCK) THEN RETURN Errr("syntax error")
00654   IF *Tk <> TOK_WORD THEN RETURN Errr("name expected")
00655   NamTok = Tk : SKIP
00656   IF *Tk <> TOK_BROPN THEN RETURN Errr("'()' expected")
00657   ParTok = Tk
00658   skipOverBrclo()
00659   Emit->Defi_(@THIS)
00660   RETURN MSG_ERROR
00661 END FUNCTION
00662
00663
00664 /'* \brief Evaluate a \#`DEFINE` declaration
00665 \returns MSG_ERROR (or MSG_STOP on the end of the token list)
00666
00667 The pre-parser found a \#`DEFINE` line. We tokenize the line and call
00668 the emitter, or we call Errr() handler on syntax problems.
00669
00670 '/
00671 FUNCTION Parser.DEFINE_() AS INTEGER
00672   IF 3 > tokenize(TO_EOL) THEN RETURN Errr("syntax error")
00673   IF *Tk <> TOK_WORD THEN RETURN Errr("name expected")
00674   NamTok = Tk : SKIP
00675   IF *Tk = TOK_BROPN ANDALSO Tk[1] = NamTok[1] + NamTok[2] _
00676     THEN ParTok = Tk : skipOverBrclo() _
00677     ELSE ParTok = 0
00678   DivTok = Tk
00679   Tk = EndTok - 3
00680   Emit->Defi_(@THIS)
00681   RETURN MSG_ERROR
00682 END FUNCTION
00683
00684
00685 /'* \brief Emit an error message
00686 \param E the reason for the error message
00687 \returns MSG_ERROR (and MSG_STOP at end of file)
00688
00689 Create an error message and call the error function of the emitter.
00690 It depends on the emitter if and where the error is shown.
00691
00692 '/
00693 FUNCTION Parser.Errr(BYREF E AS STRING) AS INTEGER
00694   VAR z = LineNo
00695   IF LevelCount THEN '                          adapt line number counter
00696     FOR i AS INTEGER = Tk[1] TO EndTok[1]
00697       IF Buf[i] = ASC(!"\n") THEN z -= 1
00698     NEXT
00699   END IF
00700   ErrMsg = "-error(" & z & "): " & E ' & _
00701            '"! (... " & MID(Buf, IIF(Po > 20, Po - 40, 1), 20)
00702   SELECT CASE AS CONST *StaTok
00703   CASE TOK_DIM  : ErrMsg &= " (DIM)"
00704   CASE TOK_RDIM : ErrMsg &= " (REDIM)"
00705   CASE TOK_VAR  : ErrMsg &= " (VAR)"
00706   CASE TOK_CONS : ErrMsg &= " (CONST)"
```

```
00707   CASE TOK_STAT : ErrMsg &= " (STATIC)"
00708   CASE TOK_COMM : ErrMsg &= " (COMMON)"
00709   CASE TOK_EXRN : ErrMsg &= " (EXTERN)"
00710   CASE TOK_TYPE : ErrMsg &= " (TYPE)"
00711   CASE TOK_CLAS : ErrMsg &= " (CLASS)"
00712   CASE TOK_SUB  : ErrMsg &= " (SUB)"
00713   CASE TOK_FUNC : ErrMsg &= " (FUNCTION)"
00714   CASE TOK_PROP : ErrMsg &= " (PROPERTY)"
00715   CASE TOK_CTOR : ErrMsg &= " (CONSTRUCTOR)"
00716   CASE TOK_DTOR : ErrMsg &= " (DESTRUCTOR)"
00717   CASE TOK_NAMS : ErrMsg &= " (NAMESPACE)"
00718   CASE TOK_SCOP : ErrMsg &= " (SCOPE)"
00719   CASE TOK_ENUM : ErrMsg &= " (ENUM)"
00720   CASE TOK_UNIO : ErrMsg &= " (UNION)"
00721   CASE TOK_DECL : ErrMsg &= " (DECLARE)"
00722   CASE TOK_DEFI : ErrMsg &= " (#DEFINE)"
00723   CASE TOK_MACR : ErrMsg &= " (#MACRO)"
00724   CASE ELSE : ErrMsg &= " (???)"
00725   END SELECT
00726   Emit->Error_(@THIS) : ErrMsg = ""
00727
00728   RETURN IIF(Buf[Po] = 0, MSG_STOP, MSG_ERROR)
00729 END FUNCTION
00730
00731
00732 /'*
00733 \brief Check word at current parser position
00734 \returns: The token for the word at current position
00735
00736 This function checks the word at the current parser position. In
00737 case of a keyword the matching token gets returned. Otherwise \ref
00738 Parser::TOK_WORD gets returned.
00739
00740 '/
00741 FUNCTION Parser.getToken() AS INTEGER
00742   SELECT CASE AS CONST Buf[*A]
00743   CASE ASC("A"), ASC("a")
00744     SELECT CASE USubStr
00745     CASE "AS" : RETURN TOK_AS
00746     CASE "ALIAS" : RETURN TOK_ALIA
00747     CASE "ABSTRACT" : RETURN TOK_ABST
00748     END SELECT
00749   CASE ASC("B"), ASC("b")
00750     SELECT CASE USubStr
00751     CASE "BYTE" : RETURN TOK_BYTE
00752     CASE "BYREF" : RETURN TOK_BYRE
00753     CASE "BYVAL" : RETURN TOK_BYVA
00754     END SELECT
00755   CASE ASC("C"), ASC("c")
00756     SELECT CASE USubStr
00757     CASE "CDECL" : RETURN TOK_CDEC
00758     CASE "CLASS" : RETURN TOK_CLAS
00759     CASE "CONST" : RETURN TOK_CONS
00760     CASE "COMMON" : RETURN TOK_COMM
00761     CASE "CONSTRUCTOR" : RETURN TOK_CTOR
00762     END SELECT
00763   CASE ASC("D"), ASC("d")
00764     SELECT CASE USubStr
00765     CASE "DIM" : RETURN TOK_DIM
00766     CASE "DOUBLE" : RETURN TOK_DOUB
00767     CASE "DEFINE" : IF ToLast = TOK_LATTE THEN RETURN TOK_DEFI
00768     CASE "DECLARE" : RETURN TOK_DECL
00769     CASE "DESTRUCTOR" : RETURN TOK_DTOR
00770     END SELECT
00771   CASE ASC("E"), ASC("e")
00772     SELECT CASE USubStr
00773     CASE "END" : RETURN TOK_END
00774     CASE "ENUM" : RETURN TOK_ENUM
00775     CASE "EXTERN" : RETURN TOK_EXRN
00776     CASE "EXPORT" : RETURN TOK_EXPO
00777     CASE "EXTENDS" : RETURN TOK_EXDS
00778     CASE "ENDMACRO" : IF ToLast = TOK_LATTE THEN RETURN TOK_EMAC
00779     END SELECT
00780   CASE ASC("F"), ASC("f")
00781     SELECT CASE USubStr
00782     CASE "FUNCTION" :  RETURN TOK_FUNC
00783     CASE "FIELD" : RETURN TOK_FILD
00784     END SELECT
00785   CASE ASC("I"), ASC("i")
00786     SELECT CASE USubStr
00787     CASE "INCLUDE" : IF ToLast = TOK_LATTE THEN RETURN TOK_INCL
00788     CASE "INTEGER" : RETURN TOK_INT
00789     END SELECT
00790   CASE ASC("L"), ASC("l")
00791     SELECT CASE USubStr
00792     CASE "LIB" : RETURN TOK_LIB
00793     CASE "LONG" : RETURN TOK_LONG
```

```
00794     CASE "LONGINT" : RETURN TOK_LINT
00795     END SELECT
00796   CASE ASC("M"), ASC("m") : IF USubStr = "MACRO" THEN IF ToLast = TOK_LATTE THEN RETURN TOK_MACR
00797   CASE ASC("N"), ASC("n") : IF USubStr = "NAMESPACE" THEN RETURN TOK_NAMS
00798   CASE ASC("O"), ASC("o")
00799     SELECT CASE USubStr
00800     CASE "ONCE" : RETURN TOK_ONCE
00801     CASE "OPERATOR" : RETURN TOK_OPER
00802     CASE "OVERLOAD" : RETURN TOK_OVER
00803     END SELECT
00804   CASE ASC("P"), ASC("p")
00805     SELECT CASE USubStr
00806     CASE "PTR" : RETURN TOK_PTR
00807     CASE "POINTER" : RETURN TOK_PTR
00808     CASE "PASCAL" : RETURN TOK_PASC
00809     CASE "PUBLIC" : RETURN TOK_PUBL
00810     CASE "PRIVATE" : RETURN TOK_PRIV
00811     CASE "PROPERTY" : RETURN TOK_PROP
00812     CASE "PROTECTED" : RETURN TOK_PROT
00813     END SELECT
00814   CASE ASC("R"), ASC("r") : IF USubStr = "REDIM" THEN RETURN TOK_RDIM
00815   CASE ASC("S"), ASC("s")
00816     SELECT CASE USubStr
00817     CASE "SUB" : RETURN TOK_SUB
00818     CASE "SCOPE" : RETURN TOK_SCOP
00819     CASE "SHORT" : RETURN TOK_SHOR
00820     CASE "SINGLE" : RETURN TOK_SING
00821     CASE "SHARED" : RETURN TOK_SHAR
00822     CASE "STRING" : RETURN TOK_STRI
00823     CASE "STATIC" : RETURN TOK_STAT
00824     CASE "STDCALL" : RETURN TOK_STCL
00825     END SELECT
00826   CASE ASC("T"), ASC("t") : IF USubStr = "TYPE" THEN RETURN TOK_TYPE
00827   CASE ASC("U"), ASC("u")
00828     SELECT CASE USubStr
00829     CASE "UNION" : RETURN TOK_UNIO
00830     CASE "UBYTE" : RETURN TOK_UBYT
00831     CASE "ULONG" : RETURN TOK_ULNG
00832     CASE "ULONGINT" : RETURN TOK_ULIN
00833     CASE "UINTEGER" : RETURN TOK_UINT
00834     CASE "USHORT" : RETURN TOK_USHO
00835     END SELECT
00836   CASE ASC("V"), ASC("v")
00837     SELECT CASE USubStr
00838     CASE "VAR" : RETURN TOK_VAR
00839     CASE "VIRTUAL" : RETURN TOK_VIRT
00840     END SELECT
00841   CASE ASC("W"), ASC("w") : IF USubStr = "WSTRING" THEN RETURN TOK_WSTR
00842   CASE ASC("Z"), ASC("z") : IF USubStr = "ZSTRING" THEN RETURN TOK_ZSTR
00843   END SELECT : RETURN TOK_WORD
00844 END FUNCTION
00845
00846
00847 /'* \brief Find the end of a Quote
00848
00849 This snippet is used to find the end of a quoted string. It checks
00850 if the string uses escape sequences and evaluates '\\"' . It stops at
00851 the last double quote (if Buf[Po] isn't ASC(!"\\"") then the end of
00852 the buffer is reached).
00853
00854 '/
00855 #MACRO SCAN_QUOTE(Buf,Po)
00856   VAR esc = IIF(Buf[Po - 1] = ASC("!"), 1, 0)
00857   DO
00858     Po += 1
00859     SELECT CASE AS CONST Buf[Po]
00860     CASE 0, ASC(!"\n") : Po -= 1 : EXIT DO
00861     CASE ASC("\") : IF esc THEN Po += 1
00862     CASE ASC(""") : IF Buf[Po + 1] = ASC(""") THEN Po += 1 ELSE EXIT DO
00863     END SELECT
00864   LOOP
00865 #ENDMACRO
00866
00867
00868 /'* \brief Find the end of a line end comment
00869
00870 This snippet is used to find the end of a line end comment. It
00871 checks for a ASC(!"\n") and evaluates line concatenations ( _ ) on
00872 the way. It stops at the line end (if Buf[Po] isn't ASC(!"\n") then
00873 the end of the buffer is reached).
00874
00875 '/
00876 #MACRO SCAN_SL_COMM(Buf,Po)
00877   DO
00878     Po += 1
00879     SELECT CASE AS CONST Buf[Po]
00880     CASE 0, ASC(!"\n") : EXIT DO
```

```
00881     END SELECT
00882   LOOP
00883 #ENDMACRO
00884
00885
00886 /'* \brief Find the end of a multi line comment
00887
00888 This snippet is used to find the end of a multi line comment block.
00889 It stops at the characters ' and / (if Buf[Po] isn't ASC("/") then the
00890 end of the buffer is reached).
00891
00892 '/
00893 #MACRO SCAN_ML_COMM()
00894   Po += 2
00895   DO
00896     SELECT CASE AS CONST Buf[Po]
00897     CASE 0 : EXIT DO
00898     CASE ASC(!"\n") : LineNo += 1
00899     CASE ASC("'")
00900       SELECT CASE AS CONST Buf[Po + 1]
00901       CASE 0 : EXIT DO
00902       CASE ASC("/") : Po += 1 : EXIT DO
00903       END SELECT
00904     END SELECT : Po += 1
00905   LOOP
00906 #ENDMACRO
00907
00908
00909 /'* \brief Find the end of a word
00910
00911 This snippet is used to find the end of a word in the input buffer.
00912 Po is at the first non-word character when done.
00913
00914 '/
00915 #MACRO SCAN_WORD()
00916   *A = Po
00917   Po += 1
00918   DO
00919     SELECT CASE AS CONST Buf[Po]
00920     CASE ASC("0") TO ASC("9"), _
00921          ASC("A") TO ASC("Z"), _
00922          ASC("a") TO ASC("z"), ASC("_") : Po += 1
00923     CASE ELSE : EXIT DO
00924     END SELECT
00925   LOOP
00926 #ENDMACRO
00927
00928 '* snippet to set the current token
00929 #DEFINE SETOK(_T_,_A_,_L_) Tok &= MKI(_T_) & MKI(_A_) & MKI(_L_) : ToLast = _T_
00930
00931 '* snippet to exit in case of end of file
00932 #DEFINE EXIT_STOP  SETOK(MSG_STOP, Po, 1) : Po -= 1 : EXIT DO
00933
00934 /'* \brief Parse a relevant construct, create a token list
00935 \param Stop_ the condition where to end parsing
00936 \returns the length of the token list in byte (at least 8)
00937
00938 Start at the current position of the input buffer to check each
00939 character. Sort the input in to a token list. The token list
00940 contains three values in each entry: the type of the input, its
00941 start and its length.
00942
00943 The end of this process gets specified by the Stop_ parameter. It's
00944 one of the #EoS_Modi enumerators.
00945
00946 '/
00947 FUNCTION Parser.tokenize(BYVAL Stop_ AS EoS_Modi) AS INTEGER
00948   VAR coca = 0, endcount = 0, newblock = 0 ', currTok = MSG_ERROR, fl_end = 0
00949   VAR tok_begin = LEN(Tok) SHR 2
00950   ToLast = IIF(*StaTok = TOK_TYPE, TOK_EOS, MSG_ERROR)
00951   A += 3 : L += 3
00952   DO
00953     SELECT CASE AS CONST Buf[Po]
00954     CASE 0 : EXIT_STOP
00955     CASE ASC(!"\n") : LineNo += 1 : IF coca THEN coca = 0 : EXIT SELECT
00956       IF ToLast = TOK_EOS THEN EXIT SELECT
00957       SETOK(TOK_EOS, Po, 1) : IF Stop_ >= TO_EOL       THEN EXIT DO
00958       IF newblock THEN endcount += 1 : newblock = 0
00959     CASE   ASC(":")
00960       SETOK(TOK_EOS, Po, 1)    : IF Stop_ >= TO_COLON   THEN EXIT DO
00961       IF newblock THEN endcount += 1 : newblock = 0
00962     CASE   ASC(",") : SETOK(TOK_COMMA, Po, 1)
00963     CASE   ASC(".")
00964       IF Buf[Po + 1] <> ASC(".") ORELSE Buf[Po + 2] <> ASC(".") ORELSE Buf[Po + 3] = ASC(".") _
00965         THEN SETOK(TOK_DOT, Po, 1) _
00966         ELSE SETOK(TOK_TRIDO, Po, 3) : Po += 3        : CONTINUE DO
00967     CASE   ASC("=") : SETOK(TOK_EQUAL, Po, 1)
```

```
00968     CASE   ASC("{"), ASC("[") : SETOK(TOK_KLOPN, Po, 1)
00969     CASE   ASC("}"), ASC("]") : SETOK(TOK_KLCLO, Po, 1)
00970     CASE   ASC("(")           : SETOK(TOK_BROPN, Po, 1)
00971     CASE   ASC(")")           : SETOK(TOK_BRCLO, Po, 1)
00972     CASE   ASC("#") : SETOK(TOK_LATTE, Po, 1)
00973
00974     CASE ASC("A") TO ASC("Z"), ASC("a") TO ASC("z"), ASC("_")
00975       SCAN_WORD()
00976       *L = Po - *A : IF *L = 1 ANDALSO Buf[*A] = ASC("_") THEN coca = *A : CONTINUE DO
00977
00978       VAR pretok = ToLast
00979       SETOK(getToken(), *A, *L)
00980       newblock = 0
00981       IF pretok = TOK_END THEN
00982         SELECT CASE AS CONST *StaTok
00983         CASE TOK_SUB, TOK_FUNC, TOK_PROP, TOK_OPER, TOK_CTOR, TOK_DTOR, TOK_ENUM
00984           IF ToLast = *StaTok THEN Stop_ = TO_COLON
00985           CONTINUE DO
00986         CASE TOK_UNIO, TOK_TYPE, TOK_CLAS
00987           IF ToLast <> *StaTok THEN CONTINUE DO
00988         CASE ELSE : CONTINUE DO
00989         END SELECT
00990         endcount -= 1 : IF endcount < 0 THEN Stop_ = TO_COLON
00991       ELSE
00992         SELECT CASE AS CONST *StaTok
00993         CASE TOK_UNIO, TOK_TYPE, TOK_CLAS
00994           IF pretok <> TOK_EOS THEN CONTINUE DO
00995           IF ToLast = *StaTok THEN newblock = 1
00996         CASE TOK_MACR : IF ToLast = TOK_EMAC THEN Stop_ = TO_COLON
00997         END SELECT
00998       END IF : CONTINUE DO
00999
01000     CASE ASC("""")
01001       *A = Po
01002       SCAN_QUOTE(Buf,Po)
01003       SETOK(TOK_QUOTE, *A, Po - *A + 1)
01004       IF Buf[Po] <> ASC("""") THEN EXIT_STOP
01005     CASE ASC("'")
01006       SCAN_SL_COMM(Buf,Po)
01007       CONTINUE DO
01008     CASE ASC("/") : IF Buf[Po + 1] <> ASC("'") THEN   EXIT SELECT
01009       SCAN_ML_COMM()
01010       IF Buf[Po] <> ASC("/") THEN EXIT_STOP
01011     END SELECT : Po += 1
01012   LOOP : A -= 3 : L -= 3 : Po += 1
01013
01014   Tk1 = CAST(INTEGER PTR, SADD(Tok))
01015   EndTok = Tk1 + (LEN(Tok) SHR 2) - 3
01016   Tk = Tk1 + tok_begin
01017   RETURN EndTok - Tk
01018 END FUNCTION
01019
01020
01021 /'* \brief Pre-parse all FB source code, search relevant constructs
01022 \returns the length of the output code, if any
01023
01024 Search the input buffer for a relevant construct. Start detailed
01025 parsing on each relevant construct. Otherways skip the current line.
01026 Export comments on the way.
01027
01028 '/
01029 SUB Parser.pre_parse()
01030   *L = 0
01031   Po = 0
01032   ToLast = MSG_ERROR
01033   LineNo = 1
01034   Tok = ""
01035   Emit->Init_(@THIS)
01036   DO
01037     SELECT CASE AS CONST Buf[Po] '                              search word
01038     CASE 0 : EXIT DO
01039     CASE ASC(!"\n") : *L = 0 : LineNo += 1
01040     CASE   ASC(":") : *L = 0 : ToLast = 0
01041     CASE   ASC("#") : ToLast = TOK_LATTE
01042     CASE ASC("""")
01043       SCAN_QUOTE(Buf,Po)
01044       ToLast = TOK_QUOTE : IF Buf[Po] <> ASC("""") THEN EXIT DO
01045     CASE ASC("'")
01046       SCAN_SL_COMM(Buf,Po)
01047       CONTINUE DO
01048     CASE ASC("/") : IF Buf[Po + 1] <> ASC("'") THEN EXIT SELECT
01049       SCAN_ML_COMM()
01050       IF Buf[Po] <> ASC(!"/") THEN EXIT DO
01051
01052     CASE ASC("_"), ASC("A") TO ASC("Z"), ASC("a") TO ASC("z")
01053       SCAN_WORD() :  IF *L THEN ToLast = 0 : CONTINUE DO ' skip word, if not first
01054       *L = Po - *A : IF *L = 1 ANDALSO Buf[*A] = ASC("_") THEN ToLast = 0 : *L = 0 : CONTINUE DO
```

```
01055
01056       ListCount = 0
01057       LevelCount = 0
01058       *StaTok = getToken()
01059       SELECT CASE AS CONST *StaTok
01060       CASE TOK_SUB, TOK_FUNC, TOK_PROP, TOK_OPER, TOK_CTOR, TOK_DTOR
01061                       IF FUNCTION_() = MSG_STOP THEN EXIT DO
01062       CASE TOK_DIM, TOK_RDIM, TOK_VAR, TOK_CONS, TOK_COMM, TOK_EXRN, TOK_EXPO, TOK_STAT
01063                       IF      VAR_() = MSG_STOP THEN EXIT DO
01064       CASE TOK_TYPE, TOK_CLAS
01065                       IF     TYPE_() = MSG_STOP THEN EXIT DO
01066       CASE TOK_UNIO : IF    UNION_() = MSG_STOP THEN EXIT DO
01067       CASE TOK_ENUM : IF     ENUM_() = MSG_STOP THEN EXIT DO
01068       CASE TOK_DECL : IF  DECLARE_() = MSG_STOP THEN EXIT DO
01069       CASE TOK_DEFI : IF   DEFINE_() = MSG_STOP THEN EXIT DO
01070       CASE TOK_MACR : IF    MACRO_() = MSG_STOP THEN EXIT DO
01071       CASE TOK_INCL : IF  INCLUDE_() = MSG_STOP THEN EXIT DO
01072       CASE TOK_PUBL, TOK_PRIV
01073         SETOK(*StaTok, *A, *L) : *L = 0 : CONTINUE DO
01074       'CASE TOK_NAMS : ' !!! ToDo
01075       'CASE TOK_SCOP : ' !!! ToDo
01076       CASE ELSE :  ToLast = *StaTok : CONTINUE DO
01077       END SELECT : ToLast = *StaTok : Tok = "" : *L = 0 : CONTINUE DO
01078     END SELECT : Po += 1
01079   LOOP : Emit->Exit_(@THIS)
01080 END SUB
01081
01082
01083 /'* \brief Read a buffer from a file and parse
01084 \param File the name of the file to translate
01085 \param Tree if to follow source tree \#`INCLUDE`
01086 \returns the translated code (if any)
01087
01088 Read a file in to input buffer. Start detailed parsing on each
01089 relevant construct. Otherways skip the current line. Export comments
01090 on the way. Do nothing if file doesn't exist or isn't readable.
01091
01092 '/
01093 SUB Parser.File_(BYREF File AS STRING, BYVAL Tree AS INTEGER)
01094   VAR fnr = FREEFILE
01095   IF OPEN(File FOR INPUT AS #fnr) _
01096     THEN ErrMsg = "couldn't read file """ & File & """ " &_
01097                   "(ERR = " & ERR & "), currdir=" & curdir : EXIT SUB
01098   Buf = STRING(LOF(fnr), 0)
01099   GET #fnr, , Buf
01100   CLOSE #fnr
01101
01102   InTree = Tree
01103   Fin = LEN(Buf) - 1
01104   IF OPT->InTree THEN InPath = left(File, instrrev(File, SLASH))
01105   pre_parse()
01106   ErrMsg = "done"
01107 END SUB
01108
01109
01110 /'* \brief Read a buffer from pipe STDIN and parse
01111 \returns the translated code (or an informal text)
01112
01113 Get all characters from STDIN. Start the parsing process. If there is
01114 no input (an empty line) then call function \ref EmitterIF::Empty_().
01115 (Useful for generating file templates in mode `--geany-mode`.)
01116
01117 '/
01118 SUB Parser.StdIn()
01119   Buf = ""
01120   VAR fnr = FREEFILE
01121   OPEN CONS FOR INPUT AS #fnr
01122     WHILE NOT EOF(fnr)
01123       DIM AS UBYTE z
01124       GET #fnr, , z
01125       Buf &= CHR(z)
01126     WEND
01127   CLOSE #fnr
01128
01129   IF LEN(Buf) < 3 THEN Emit->Empty_(@THIS) : EXIT SUB
01130
01131   Fin = LEN(Buf) - 1
01132   pre_parse() : IF Tk1 THEN EXIT SUB
01133
01134   Code(  "'                    " & PROG_NAME & ": no --geany-mode output:" & _
01135     NL & "'                       select either a line" & _
01136     NL & "'         DIM, COMMON, CONST, EXTERN, STATIC, DECLARE, #DEFINE" & _
01137     NL & "'                              or a block" & _
01138     NL & "'                        ENUM, UNION, TYPE, #MACRO" & _
01139     NL & "'                                 or a" & _
01140     NL & "'                       SUB, FUNCTION or PROPERTY" & _
01141     NL & "'                             declaration or" & _
```

```
01142     NL & "'                   place the cursor in an empty line" & _
01143     NL)
01144 END SUB
01145
01146
01147 /'* \brief write a piece of output (for external emitters only)
01148 \param T the text to write
01149
01150 External emitters cannot use the streams opened in the main program
01151 directly. They have to send text to this procedure to use the standard
01152 output stream.
01153
01154 '/
01155 SUB Parser.writeOut(BYREF T AS STRING) EXPORT
01156   Code(T)
01157 END SUB
01158
01159
01160 /'* \brief The current token
01161 \returns the token the parser currently stopping at
01162
01163 This property returns the parser token (where the parser currently
01164 stops).
01165
01166 '/
01167 PROPERTY Parser.CurTok() AS INTEGER PTR EXPORT
01168   RETURN Tk
01169 END PROPERTY
01170
01171 /'* \brief The initialization of a bitfield
01172 \returns the bitfield initialization
01173
01174 This property returns the initialization of a bitfield in a TYPE / UNION
01175 block. The size of the bitfield may either be an integer number or a
01176 macro (= TOK_WORD).
01177
01178 '/
01179 PROPERTY Parser.BitIni() AS STRING EXPORT
01180   IF BitTok[3] = TOK_WORD THEN RETURN ": " & SubStr(BitTok + 3)
01181   VAR a = BitTok[1] + 1, l = 0
01182   FOR i AS INTEGER = a TO BitTok[4] - 1
01183     SELECT CASE AS CONST Buf[i]
01184     CASE ASC(" "), ASC(!"\t"), ASC(!"\v") : IF l THEN EXIT FOR
01185     CASE ASC("0") TO ASC("9") : IF l THEN l += 1 ELSE a = i : l = 1
01186     CASE ELSE : EXIT FOR
01187     END SELECT
01188   NEXT : RETURN ": " & MID(Buf, a + 1, l)
01189 END PROPERTY
01190
01191 '/'* \brief The dimension of a variable
01192 '\returns the name including round or squared brackets
01193
01194 'This property reads the dimension of a variable from the input buffer.
01195
01196 ''/
01197 'PROPERTY Parser.Bracket(BYVAL T AS INTEGER PTR) AS STRING EXPORT
01198   'VAR kl = 0, p = T[1], l = 0
01199   'WHILE T < EndTok
01200     'SELECT CASE AS CONST *T
01201     'CASE TOK_COMMA : IF kl <= 0 THEN l = T[1] - p : EXIT WHILE
01202     'CASE TOK_KLOPN, TOK_BROPN : kl += 1
01203     'CASE TOK_KLCLO, TOK_BRCLO : kl -= 1
01204       'IF kl <= 0 THEN l = T[1] - p  + 1 : EXIT WHILE
01205     'END SELECT : T += 3
01206   'WEND : RETURN MID(Buf, p + 1, l)
01207 'END PROPERTY
01208
01209 '/'* \brief The variable initialization
01210 '\returns the initialiser
01211
01212 'This property reads the initialization of a variable from the input
01213 'buffer.
01214
01215 ''/
01216 'PROPERTY Parser.VarIni() AS STRING EXPORT
01217   'VAR kl = 0, i = IniTok[1], p = i
01218   'DO
01219     'i += 1
01220     'SELECT CASE AS CONST Buf[i]
01221     'CASE 0, ASC(!"\n"), ASC(!"\r"), ASC("'") : EXIT DO
01222     '''CASE 0    : EXIT DO
01223     '''CASE ASC("'")    : IF kl <= 0 THEN EXIT DO
01224     '''CASE ASC(!"\n") : LineNo += 1   : IF kl <= 0 THEN EXIT DO
01225     'CASE ASC("_") : IF Buf[i + 1] < ASC("0") THEN EXIT DO
01226     '''CASE ASC("_") : IF Buf[i + 1] < ASC("0") THEN IF kl <= 0 THEN EXIT DO
01227     'CASE ASC("""")
01228       'VAR esc = IIF(Buf[i - 1] = ASC("!"), 1, 0)
```

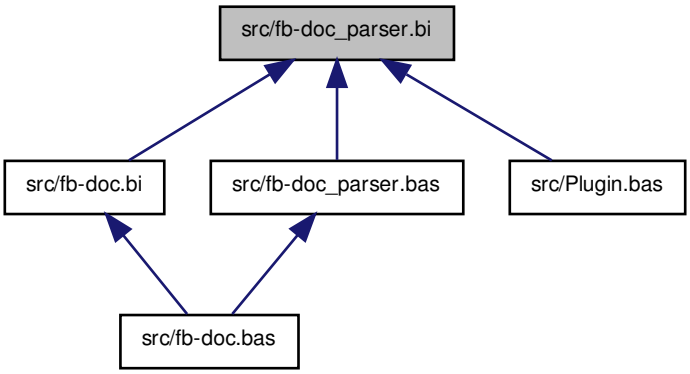
```
01229       'DO
01230         'i += 1
01231         'SELECT CASE AS CONST Buf[i]
01232         'CASE 0, ASC(!"\n") : i -= 1 : EXIT DO
01233         'CASE ASC("\") : IF esc THEN i += 1
01234         'CASE ASC(""") : IF Buf[i + 1] = ASC(""") THEN i += 1 ELSE EXIT DO
01235         'END SELECT
01236       'LOOP
01237     'CASE ASC("("), ASC("{"), ASC("[") : kl += 1
01238     'CASE ASC(")"), ASC("}"), ASC("]") : kl -= 1 : IF kl <  0 THEN EXIT DO
01239     'CASE ASC(",")                               : IF kl <= 0 THEN EXIT DO
01240     'END SELECT
01241   'LOOP : RETURN " " & RTRIM(MID(Buf, p + 1, i - p), ANY !" \t\v")
01242 'END PROPERTY
01243
01244 /'* \brief Context of current token
01245 \returns the context of the current token
01246
01247 This property returns the context of the current token from the
01248 input buffer.
01249
01250 '/
01251 PROPERTY Parser.SubStr() AS STRING EXPORT
01252   IF 0 = Tk THEN RETURN " ?? 0 ?? "
01253   RETURN MID(Buf, Tk[1] + 1, Tk[2])
01254 END PROPERTY
01255
01256 /'* \brief Check current token position, extract word if string type
01257 \param T The token to read
01258 \returns the context of the token T
01259
01260 This property returns the context of the token T from the
01261 input buffer.
01262
01263 '/
01264 PROPERTY Parser.SubStr(BYVAL T AS INTEGER PTR) AS STRING EXPORT
01265   IF 0 = T THEN RETURN " ?? 0 ?? "
01266   RETURN MID(Buf, T[1] + 1, T[2])
01267 END PROPERTY
01268
01269
01270 /'* \brief Check current token position, extract word if string type
01271 \returns the current word in upper case
01272
01273 This property returns the context of the current parser position in the
01274 input buffer in upper case characters.
01275
01276 '/
01277 PROPERTY Parser.USubStr() AS STRING
01278   RETURN UCASE(MID(Buf, *A + 1, Po - *A))
01279 END PROPERTY
01280
01281
01282 /'* \brief start a new parser to \#`INCLUDE` a file
01283 \param N the (path, if any, and) file name
01284
01285 This procedure is used by the emitter handlers for \#`INCLUDE`
01286 statements. It checks if the file hasn't been done already and can get
01287 opened. If one of these fails a message gets sent to STDERR and the
01288 file gets skipped.
01289
01290 Otherwise a new parser gets started to operate on that file.
01291
01292 '/
01293 SUB Parser.Include(BYVAL N AS STRING)
01294 with *OPT
01295   IF .RunMode = .GEANY_MODE THEN EXIT SUB
01296
01297   VAR i = INSTRREV(N, SLASH)
01298   VAR fnam = .addPath(InPath, LEFT(N, i)) & MID(N, i + 1)
01299   MSG_LINE(fnam)
01300   IF DivTok ANDALSO INSTR(.FileIncl, !"\n" & fnam & !"\r") THEN _
01301     MSG_END("skipped (already done)") : EXIT SUB
01302
01303   VAR fnr = FREEFILE
01304   IF OPEN(fnam FOR INPUT AS #fnr) THEN _
01305     MSG_END("skipped (couldn't open)") : EXIT SUB
01306   CLOSE #fnr
01307   .FileIncl &= !"\n" & fnam & !"\r"
01308
01309   VAR pars_old = .Pars : .Pars = NEW Parser(.EmitIF)
01310   UserTok = pars_old->UserTok
01311
01312   MSG_END("working ...")
01313   .Level += 1
01314   .doFile(fnam)
01315   .Level -= 1
```

```
01316
01317   SELECT CASE AS CONST .RunMode
01318   CASE .LIST_MODE, .FILE_MODE
01319   CASE ELSE : MSG_LINE(fnam) : MSG_END("done")
01320   END SELECT
01321
01322   DELETE .Pars : .Pars = pars_old
01323 END with
01324 END SUB
01325
```

15.33 src/fb-doc_parser.bi File Reference

Header file for the Parser class.

This graph shows which files directly or indirectly include this file:



Classes

- class Parser

  The parser.

Macros

- #define Code(_T_) /∗ P->writeOut(_T_) '∗< Convenience macro for output (plugin) ∗/
- #define Code(_T_) /∗ PRINT #OPT->Ocha, _T_; '∗< Convenience macro for output (fb-doc) ∗/

Variables

- const VAR SLASH = "/"

  separator for folders (unix version)

- const VAR NL = !"\n"

  separator for lines (unix version)

15.33.1 Detailed Description

Header file for the Parser class. This file contains the declaration of the Parser, a class for loading and scanning FB source code. It calls the matching functions in the EmitterIF.

Definition in file fb-doc_parser.bi.

15.33.2 Macro Definition Documentation

15.33.2.1 #define Code(_T_) /* P->writeOut(_T_) '*< Convenience macro for output (plugin) */

Definition at line 13 of file fb-doc_parser.bi.

15.33.2.2 #define Code(_T_) /* PRINT #OPT->Ocha, _T_; '*< Convenience macro for output (fb-doc) */

Definition at line 13 of file fb-doc_parser.bi.

15.33.3 Variable Documentation

15.33.3.1 const VAR SLASH = "/"

separator for folders (unix version)

Definition at line 18 of file fb-doc_parser.bi.

15.33.3.2 const VAR NL = !"\n"

separator for lines (unix version)

Definition at line 19 of file fb-doc_parser.bi.

15.34 fb-doc_parser.bi

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'* \file fb-doc_parser.bi
00002 \brief Header file for the #Parser class
00003
00004 This file contains the declaration of the \ref Parser, a class for
00005 loading and scanning FB source code. It calls the matching functions in
00006 the \ref EmitterIF.
00007
00008 '/
00009
00010 #IF __FB_OUT_DLL__
00011 #DEFINE Code(_T_) P->writeOut(_T_) '*< Convenience macro for output (plugin)
00012 #ELSE
00013 #DEFINE Code(_T_) PRINT #OPT->Ocha, _T_; '*< Convenience macro for output (fb-doc)
00014 #ENDIF
00015
00016 #IFDEF __FB_UNIX__
00017  CONST _
00018    SLASH = "/" _ '*< separator for folders (unix version)
00019     , NL = !"\n" '*< separator for lines (unix version)
00020 #ELSE
00021 '&/*
00022  CONST _
00023    SLASH = "\" _
00024     , NL = !"\r\n"
00025 '&*/
00026 #ENDIF
00027
00028 /'* \brief The parser
00029
00030 Class to handle FreeBasic source code. A Parser allways work on
00031 exactly one input stream, comming from a file or from STDIN. The
```

```
00032 Parser does
00033
00034  - read the source from an input channel (see \ref StdIn(), \ref File_())
00035  - call function \ref EmitterIF::Init_() before parsing
00036  - pre-parse the source for relavant code fractions (see \ref pre_parse())
00037    and call the comments emitter-handler on the way.
00038  - fine-scan any relavant code fractions (see \ref tokenize()) and call
00039    the matching emitter-handler for this code (done by the private
00040    functions ie like \ref DEFINE_(), \ref FUNCTION_(), \ref VAR_(), ...)
00041  - provide functions to extract elements form the source (like
00042    \ref SubStr(), \ref CurTok(), \ref BitIni(), ...)
00043  - call matching emitter functions to generaate the output stream
00044  - call function \ref EmitterIF::Exit_() after parsing
00045
00046 When fb-doc follows the source tree (option `--tree`), the
00047 function \ref EmitterIF::Incl_() creates a new Parser for each file.
00048
00049 '/
00050 TYPE Parser
00051 /'* \brief The tokens used by the parser
00052
00053 Enumerators used to classify the type of a token found in the
00054 FreeBasic source code.
00055
00056 '/
00057   ENUM ParserTokens
00058     MSG_STOP = -1 '*< end of file / token list reached
00059     MSG_ERROR     '*< create error message, continue parsing
00060     TOK_EOS   '*< end of statement (either new line or `":"`)
00061     TOK_BRCLO '*< right parenthesis
00062     TOK_COMMA '*< a comma
00063
00064     TOK_KLOPN '*< left other bracket
00065     TOK_KLCLO '*< right other bracket
00066     TOK_DOT   '*< a dot
00067     TOK_TRIDO '*< three dots
00068     TOK_QUOTE '*< a string constant (including quotes)
00069     TOK_LATTE '*< the '#' character
00070     TOK_BROPN '*< left parenthesis
00071     TOK_EQUAL '*< the '=' character
00072
00073     TOK_ABST  '*< the ABSTRACT keyword
00074     TOK_ALIA  '*< the ALIAS keyword
00075     TOK_AS    '*< the AS keyword
00076     TOK_BYRE  '*< the BYREF keyword
00077     TOK_BYVA  '*< the BYVAL keyword
00078     TOK_CDEC  '*< the CDECL keyword
00079     TOK_CLAS  '*< the CLASS keyword
00080     TOK_COMM  '*< the COMMON keyword
00081     TOK_CONS  '*< the CONST keyword
00082     TOK_DECL  '*< the DECLARE keyword
00083     TOK_DEFI  '*< the \#`DEFINE` keyword
00084     TOK_DIM   '*< the DIM keyword
00085     TOK_END   '*< the END keyword
00086     TOK_ENUM  '*< the ENUM keyword
00087     TOK_EXDS  '*< the EXTENDS keyword
00088     TOK_EMAC  '*< the \#`ENDMACRO` keyword
00089     TOK_EXRN  '*< the EXTERN keyword
00090     TOK_EXPO  '*< the EXPORT keyword
00091     TOK_INCL  '*< the \#`INCLUDE` keyword
00092     TOK_LIB   '*< the LIB keyword
00093     TOK_MACR  '*< the \#`MACRO` keyword
00094     TOK_NAMS  '*< the NAMESPACE keyword
00095     TOK_ONCE  '*< the ONCE keyword
00096     TOK_OVER  '*< the OVERLOAD keyword
00097     TOK_PASC  '*< the PASCAL keyword
00098     TOK_PRIV  '*< the PRIVATE keyword
00099     TOK_PROT  '*< the PROTECTED keyword
00100     TOK_RDIM  '*< the REDIM keyword
00101     TOK_PTR   '*< the PTR or POINTER keyword
00102     TOK_PUBL  '*< the PUBLIC keyword
00103     TOK_SCOP  '*< the SCOPE keyword
00104     TOK_SHAR  '*< the SHARED keyword
00105     TOK_STAT  '*< the STAvarTIC keyword
00106     TOK_STCL  '*< the STDCALL keyword
00107     TOK_TYPE  '*< the TYPE keyword
00108     TOK_UNIO  '*< the UNION keyword
00109     TOK_VAR   '*< the VAR keyword
00110     TOK_VIRT  '*< the VIRTUAL keyword
00111
00112     TOK_FUNC  '*< the FUNCTION keyword
00113     TOK_FILD  '*< the FIELD keyword
00114     TOK_OPER  '*< the OPERATOR keyword
00115     TOK_PROP  '*< the PROPERTY keyword
00116     TOK_SUB   '*< the SUB keyword
00117     TOK_CTOR  '*< the CONSTRUCTOR keyword
00118     TOK_DTOR  '*< the DESTRUCTOR keyword
```

```
00119
00120     TOK_BYTE  '*< the BYTE data type
00121     TOK_DOUB  '*< the DOUBLE data type
00122     TOK_INT   '*< the INTEGER data type
00123     TOK_LONG  '*< the LONG data type
00124     TOK_LINT  '*< the LONGINT data type
00125     TOK_STRI  '*< the STRING data type
00126     TOK_SHOR  '*< the SHORT data type
00127     TOK_SING  '*< the SINGLE data type
00128     TOK_UBYT  '*< the UBYTE data type
00129     TOK_ULNG  '*< the ULONG data type
00130     TOK_ULIN  '*< the ULONGINT data type
00131     TOK_UINT  '*< the UINTEGER data type
00132     TOK_USHO  '*< the USHORT data type
00133     TOK_WSTR  '*< the ZSTRING data type
00134     TOK_ZSTR  '*< the ZSTRING data type
00135     TOK_WORD  '*< a word
00136
00137     TOK_COMSL '*< line end comment (single line)
00138     TOK_COMML '*< multi line comment
00139   END ENUM
00140
00141 /'* \name Parser STRINGs
00142
00143 STRING variables to exchange data with th emitters.
00144
00145 \{
00146 '/
00147   AS STRING _
00148     InPath = "" _ '*< path of current of input file (option `--tree`)
00149   , Buf _         '*< the input buffer
00150   , ErrMsg _      '*< an error message
00151   , Tok _         '*< the token list (binaries)
00152   , BlockNam      '*< the name of a block (ENUM, UNION, TYPE)
00153
00154 '* \}
00155
00156
00157 /'* \name Parser token pointers
00158
00159 Pointers to the token list, used to specify the result of the
00160 parsing process. The tokens specify the construct in progress and
00161 they are used by the emitters to create their output.
00162
00163 Four main tokens specify the kind of the construct:
00164
00165  - StaTok: the start of the current construct
00166  - NamTok - subtokens: DimTok, IniTok (may be zero in parameter lists)
00167  - TypTok - subtokens: ShaTok, PtrTok, Co1Tok, Co2Tok (is zero for VAR, may be zero for CONST)
00168  - FunTok - subtokens: ParTok, AliTok, DivTok, By_Tok (is zero for variables)
00169
00170 Other tokens are only valid if the main token is not zero.
00171
00172 \{
00173 '/
00174   AS INTEGER PTR _
00175     StaTok, _  '*< the pre-parsed token
00176     NamTok, _  '*< the name token of the construct
00177     DimTok, _  '*< the token of the left parenthesis of a dimension
00178     IniTok, _  '*< the start token of an initializer ('=')
00179     BitTok, _  '*< the start token of an bitfiled declaration (':')
00180     TypTok, _  '*< the token of the type keyword
00181     ShaTok, _  '*< the token of the SHARED keyword
00182     PtrTok, _  '*< the token of the first PTR keyword
00183     Co1Tok, _  '*< the token of the first CONST keyword (if any)
00184     Co2Tok, _  '*< the token of the second CONST keyword (if any)
00185     FunTok, _  '*< type of the pre-parsed token
00186     CalTok, _  '*< the token of the calling convention keyword (if any)
00187     AliTok, _  '*< the token of the ALIAS keyword (if any)
00188     As_Tok, _  '*< the token of the AS keyword (if no SUB)
00189     ParTok, _  '*< the token of the left parenthesis of a parameter list
00190     By_Tok, _  '*< the token of the declaration specifier (BYVAL / BYREF)
00191     DivTok, _  '*< the token of different purposes like the STATIC keyword in a member function declaration (if any) or
00192     Tk1, _     '*< the first token in a statment
00193     EndTok, _   '*< the last token in the list
00194     UserTok    '*< a token for customised usage in emitter-handlers
00195
00196 '* \}
00197
00198 /'* \name Parser counters, integers and pointers
00199
00200 Diverse variables used on different events. They inform the emitters
00201 about the state of the parser.
00202
00203 \{
00204 '/
00205   AS INTEGER _
```

```
00206     Po, _          '*< the current position in the input buffer \ref Parser::Buf
00207     Fin, _         '*< the last position in the input buffer \ref Parser::Buf
00208     PtrCount, _    '*< the number of PTR keywords
00209     SrcBgn, _      '*< start position to export FB source
00210     LineNo, _      '*< the current line in the input buffer \ref Parser::Buf
00211     LevelCount, _ '*< the level in nested blocks (one based)
00212     InTree,      _ '*< flag to indicate if to follow source tree \#'INCLUDE's
00213     ListCount      '*< the current entry in a list (zero based)
00214
00215 '* \}
00216
00217   DECLARE CONSTRUCTOR(BYVAL AS EmitterIF PTR)
00218
00219 /'* \name Filehandlers
00220
00221 Filehandlers are used to load some FreeBasic source code input in to
00222 the buffer \ref Parser::Buf from different input channels (may be the
00223 STDIN pipe, a single file or all files in a folder). Afterwards the
00224 source code gets parsed and translated output created by the emitter
00225 gets returned.
00226
00227 \{
00228 '/
00229   DECLARE SUB File_(BYREF AS STRING, BYVAL AS INTEGER)
00230   DECLARE SUB StdIn()
00231   DECLARE SUB Include(BYVAL AS STRING)
00232   DECLARE STATIC SUB writeOut(BYREF AS STRING)
00233 '* \}
00234
00235 /'* \name Properties to extract original source code
00236
00237 Properties are used to extract original text from the input buffer
00238 at a certain position. This is the current position of the parser,
00239 or the position  of the given parameter.
00240
00241 \{
00242 '/
00243   DECLARE PROPERTY CurTok() AS INTEGER PTR
00244   DECLARE PROPERTY BitIni() AS STRING
00245   'DECLARE PROPERTY VarIni() AS STRING
00246   DECLARE PROPERTY SubStr() AS STRING
00247   DECLARE PROPERTY SubStr(BYVAL AS INTEGER PTR) AS STRING
00248   'DECLARE PROPERTY Bracket(BYVAL AS INTEGER PTR) AS STRING
00249 '* \}
00250
00251 /'* \name Parsers for lists and blocks
00252
00253 External demuxers are called from emitter-handlers when they sit on
00254 a list of statments, ie in an ENUM block or in a DIM statement with
00255 more than one variable.
00256
00257 \{
00258 '/
00259   DECLARE SUB parseListNam(BYVAL AS EmitFunc)
00260   DECLARE SUB parseListNamTyp(BYVAL AS EmitFunc)
00261   DECLARE SUB parseListPara(BYVAL AS EmitFunc)
00262   DECLARE SUB parseBlockEnum(BYVAL AS EmitFunc)
00263   DECLARE SUB parseBlockTyUn(BYVAL AS EmitFunc)
00264 '* \}
00265
00266 Private:
00267
00268 /'* \brief The modi for end of statement searching
00269
00270 Enumerators used in internal in the parser to specify where to stop
00271 to tokenize the input buffer
00272
00273 '/
00274   ENUM EoS_Modi
00275     TO_END_BLOCK  '*< tokenize to the end of the structure
00276     TO_END        '*< stop at the end of the token list
00277     TO_EOL        '*< stop at the next line end
00278     TO_COLON      '*< stop at the next line end or colon
00279   END ENUM
00280
00281 /'* \name Internal values
00282
00283 Variables used in pre-parsing process.
00284
00285 \{
00286 '/
00287   AS INTEGER _
00288     ToLast    '*< the type of the token before the current one
00289
00290   AS INTEGER PTR _
00291     Tk, _     '*< the current token of the parser
00292     A, _      '*< start of a word in pre-parsing
```

```
00293     L         '*< length of a word in pre-parsing
00294
00295   AS EmitterIF PTR Emit  '*< the emitter interface
00296
00297 '* \}
00298
00299 /'* \name Internal parsers
00300
00301 Functions for evaluating a construct in the FreeBasic source code after
00302 a relevant keyword was found in the pre-parsering process. After the
00303 fine-parsing process the matching emitter-handler gets called. (Or the
00304 function \ref EmitterIF::Error_() in case of a syntax problem. It's up
00305 to the emitter if and where the user sees the error message.)
00306
00307 \{
00308 '/
00309   DECLARE SUB pre_parse()
00310   DECLARE PROPERTY USubStr() AS STRING
00311   DECLARE FUNCTION getToken() AS INTEGER
00312   DECLARE FUNCTION tokenize(BYVAL AS EoS_Modi) AS INTEGER
00313   DECLARE FUNCTION Errr(BYREF AS STRING) AS INTEGER
00314
00315   DECLARE FUNCTION demuxNam(BYVAL AS INTEGER = TOK_WORD, BYVAL AS INTEGER = 0) AS INTEGER
00316   DECLARE FUNCTION demuxTyp(BYVAL AS INTEGER = 0) AS INTEGER
00317   DECLARE FUNCTION demuxDecl() AS INTEGER
00318
00319   DECLARE FUNCTION skipOverColon() AS INTEGER
00320   DECLARE FUNCTION skipOverBrclo() AS INTEGER
00321   DECLARE FUNCTION skipOverComma() AS INTEGER
00322
00323   DECLARE FUNCTION VAR_() AS INTEGER
00324   DECLARE FUNCTION TYPE_() AS INTEGER
00325   DECLARE FUNCTION ENUM_() AS INTEGER
00326   DECLARE FUNCTION UNION_() AS INTEGER
00327   DECLARE FUNCTION FUNCTION_() AS INTEGER
00328   DECLARE FUNCTION DECLARE_() AS INTEGER
00329   DECLARE FUNCTION INCLUDE_() AS INTEGER
00330   DECLARE FUNCTION DEFINE_() AS INTEGER
00331   DECLARE FUNCTION MACRO_() AS INTEGER
00332 '* \}
00333
00334 END TYPE
00335
```

15.35 src/Files.md File Reference

15.36 src/Files.md

```
00001 Files and Folders  {#pageFilesUsed}
00002 =================
00003 \tableofcontents
00004
00005 ![Files used or generated by fb-doc](Overview.png)
00006
00007 fb-doc currently uses two external files. It tries to load the
00008 context when needed and continues without the context when the files
00009 aren't present. fb-doc searches for the files in the EXEPATH (the
00010 folder of the fb-doc binary).
00011
00012 \section sectFileCsrc ../doc/c_src
00013
00014 This folder gets created by fb-doc to write the file output. It's
00015 used to collect the translations from the \em C_Source emitter. The
00016 back-end read its input from here.
00017
00018 \section sectFileLfn fb-doc.lfn
00019
00020 The suffix lfn means list of function names. This file is used in the
00021 C_Source emitter to create pseudo function calls in a function body. It
00022 gets created by the ListOfFunction emitter when using (option
00023 `--list-mode`) and it isn't used in Geany mode (option `--geany-mode`,
00024 emitters GtkDocTemplates and DoxygenTemplates).
00025
00026 Allthough it's usually created by fb-doc, it also can get
00027 manipulated by any text editor.
00028
00029 The file contains the names of all functions in the source code.
00030 Member functions of a UDT are listed as they are named in the
00031 function body (ie TypeName.FunctionName). The first line is empty.
00032 Further lines contain the names, one in each line, including the dot
00033 in case of a member function. The lines \b must be separated by a
```

```
00034 single line feed character (LINUX line end = 'CHR(10)'). So be
00035 careful when we edit on DOS / windows (ie use Geany and switch to
00036 LF line ends).
00037
00038 This file is needed when running as a filter for Doxygen to create
00039 the caller and callees graphs. Therefor the bodys of the function
00040 have to include the function calls. fb-doc checks all words in a
00041 function body against the known function names. Any match gets
00042 emitted as a pseudo function call (with empty parameter list).
00043
00044 Since in this mode Doxygen serves the file names in any order and
00045 fb-doc gets started new for each file, it doesn't know about a
00046 function declaration from file A when working on file B. Therefor the
00047 function names have to be served from an external source. They get
00048 red from this file at each program start.
00049
00050 The file offers an additional way to control the context of the caller
00051 and callees graphs. Only functions listed in this file gets included
00052 in the C source (and in the graphs).
00053
00054 When this file isn't present fb-doc continues execution without any
00055 message and the function bodys of the pseudo C code are empty. This
00056 means no caller or callees graphs will be created.
00057
00058
00059 \section sectFilePlugin External Moduls (*.so / *.dll)
00060
00061 fb-doc can use external emitter modules. These are executable
00062 binaries, compiled and linked as a library for dynamic linkage,
00063 either by using the FreeBasic compiler or any other compiler / linker
00064 combination. These files may have any name and get the system
00065 specific file name extension (= suffix) \em .so on LINUX or \em .dll
00066 on windows.
00067
00068 Such files aren't necessary for basic usage of fb-doc. They are used
00069 for external emitters to extend the fb-doc features and get loaded on
00070 demand (option `--emitter`) at run-time, see \ref pageExtend for
00071 details. So you may have one or more files with one of these
00072 suffixes in your working directory.
```

15.37 src/Installation.md File Reference

15.38 src/Installation.md

```
00001 Compiling and Installing  {#pageInstall}
00002 ======================
00003 \tableofcontents
00004
00005 Before you can start, you've to build your personal binary of fb-doc.
00006 It gets shipped as a FreeBasic source tree. Find the code in the
00007 folder *src* in your *fb-doc.zip* archive. To create your binary you
00008 need to install the [FreeBasic compiler
00009 fbc](http://www.freebasic.net/get) for your operating system. Then just
00010 extract the *fb-doc* folder to any place on your hard disk, change to
00011 the folder *src* and compile the main file *fb-doc.bas*. Ie load the
00012 file in to Geany IDE and choose menu <em>Build -> Compile</em>. Or call
00013 the Freebasic compiler at the command line (in the extracted
00014 folder *src*)
00015 ~~~{.sh}
00016 cd .../fb-doc/src
00017 fbc -w all fb-doc.bas
00018 ~~~
00019
00020 This creates an executable binary named
00021
00022 - *fb-doc* (on UNIX-like systems) or
00023 - *fb-doc.exe* (on other systems).
00024
00025 That's all you need to get started. Now you can use fb-doc and check
00026 its features, ie translate FB code to intermediate C format or generate
00027 templates.
00028
00029 Therefor you don't need a complex installation. Just place the compiled
00030 binary in any of your source folders and execute it there (see \ref
00031 pageUsage for details). Or you test your fb-doc executable on its own
00032 source, find some examples in \ref sectExaCli.
00033
00034 To generate a real documentation &mdash; as mentioned in the
00035 Introduction, fb-doc is not a complete documentation generator &mdash;
00036 you have to install an additional back-end for C source, like
00037
```

```
00038 - [gtk-doc](http://developer.gnome.org/gtk-doc-manual/stable/index.html)
00039 - [Doxygen](http://www.doxygen.org/)
00040
00041 Download one of these packages for your operating system and follow the
00042 installation instruction. Both can get extended by using further tools.
00043 Consider to install them too. (Ie the package
00044 [GraphViz](http://www.graphviz.org/) is used to generate the
00045 caller / callee graphs in this documentation.)
00046
00047 The first back-end is a good choise if you like to document a library
00048 API in the GNOME documentation style. In all other cases &mdash;
00049 especially when you have to document a program or application &mdash;
00050 the later is the better choise. Doxygen is a more modern tool, it's
00051 easier to handle, comes with a GUI front-end and has more features.
00052
00053 When you intend to use fb-doc in a regular basis and call it from
00054 different folders for several projects, it's best to install fb-doc on
00055 your system, as specified in one of the two following sections.
00056
00057 \section sectInstallLin UNIX Installation
00058
00059 It's recommended to create a link to the executable and move this link
00060 to a folder in the system `PATH`. Using a link allows
00061
00062 - recompiling of the source (ie for a new release) and also
00063 - using the fresh executable from any directory.
00064
00065 This can be done by executing in a terminal
00066 ~~~{.sh}
00067 cd .../fb-doc/src
00068 cp -l fb-doc ~/bin/fbdoc
00069 ~~~
00070
00071 Assuming that the folder `~/bin` is in your PATH you can now execute
00072 fb-doc in each folder by calling the link. Try
00073
00074 \code{.sh}
00075 cd ~
00076 fbdoc --version
00077 \endcode
00078
00079 and you should see the version information text in the terminal. This
00080 solution works on your personal user account. When you need a system
00081 wide installation that allows all users to access fb-doc, then you need
00082 admin privileges to place the link in a similar system folder, ie in like
00083 ~~~{.sh}
00084 cd .../fb-doc/src
00085 sudo cp -l fb-doc /usr/bin/fbdoc
00086 ~~~
00087
00088 \note Replace `.../` by a proper path on your system.
00089
00090
00091 \section sectInstallWoe DOS / windows Installation
00092
00093 DOS doesn't support links and on the other system links don't work at
00094 the terminal (AFAIK). So it's best to use a batch file to call the
00095 fb-doc executable.
00096
00097 Here's the recommended way
00098
00099 -# create a folder `C:\bin` (or use an existing folder for batch files in you `PATH`)
00100 -# ensure that this folder is in the system `PATH`
00101 -# create a file named `fbdoc.bat` in this path with the following context
00102
00103 ~~~{.bat}
00104 C:\YOUR\PATH\TO\fb-doc\src\fb-doc.exe %*
00105 ~~~
00106
00107 Then test the installation by executing
00108 ~~~{.bat}
00109 cd \
00110 fbdoc --version
00111 ~~~
00112
00113 and you should see the version information text in the terminal.
00114
00115 \note Replace `C:\YOUR\PATH\TO` by a proper path on your system.
00116
00117
00118 \section sectInstallGeany Geany IDE Installation
00119
00120 fb-doc can be used as a filter (= custom command) for Geany IDE. Using
00121 this feature, we can send the current selection (a text fragment) to
00122 the filter and receive the filtered output as replacement for the
00123 selected text. From a user point of view it looks like paste a new text
00124 block in to a previously selected section.
```

```
00125
00126 To get this running choose menu item *Edit->Format->Send Selection
00127 to->Set Custom Commands* and click on *Add* to get a new item. Then type
00128 ~~~{.sh}
00129 fbdoc --geany-mode
00130 ~~~
00131
00132 to use the default emitter for gtk-doc templates or select the emitter
00133 for Doxygen templates by
00134 ~~~{.txt}
00135 fbdoc --geany-mode "DoxygenTemplates"
00136 ~~~
00137
00138 To test the installation open a new editor window. Now right-click on
00139 an empty line and choose menu item *Format->Send Selection
00140 to->YOUR_NEW_COMMAND_ITEM*. The following text should appear (assuming
00141 you're using the DoxygenTemplates setting)
00142
00143 \code{.txt}
00144 /'* \file FIXME
00145 \brief FIXME
00146
00147 FIXME
00148
00149 '/
00150 \endcode
00151
00152 This is the output from the function \ref EmitterIF::Empty_() of the
00153 `DoxygenTemplates` emitter. Such a comment block is used to start any
00154 Doxygen input file.
00155
00156 If you don't get this text nor any other message in the editor then
00157 check the *status* message tab (in the notebook window at the bottom)
00158 to get a hint for solving the problem.
00159
00160 \note Instead of the right-click menu try also keybindings &mdash;
00161       `<Crtl>-[1 | 2 | 3]` is default. Find further details in [Geany
00162
       documentation](http://www.geany.org/manual/current/index.html#sending-text-through-custom-commands").
00163 \note This setting only works when fb-doc is installed in your system
00164       `PATH` (see previous sections for details). If you don't want to
00165       install fb-doc, you have to use the full path and the original
00166       name of the executable (ie like '/home/me/proj/fb-doc/src/fb-doc
00167       -g Doxy`).
```

15.39 src/Introduction.md File Reference

15.40 src/Introduction.md

```
00001 Introduction  {#pageIntro}
00002 ===========
00003 \tableofcontents
00004
00005 It's state-of-the-art in software development to write and edit the
00006 documentation context inside the source code. The programmer(s) can
00007 adapt the documentation on each improvement or bug fix at one place.
00008 All work is done in the source file(s), the documentation is placed in
00009 special comments in the source code so that the compiler doesn't see
00010 them. Beside the compiler, which extracts information for the CPU, an
00011 additional tool-chain is used to parse the special comments and the
00012 related source code to build the documentation, avoiding redundand
00013 data in separate files and keeping comments to a minimal size.
00014
00015 Powerful tool-chains (back-ends) exist for several programming
00016 languages (like C) to generate output in different formats (ie like
00017 *html*, *pdf*, *man-pages* and others). But previous to fb-doc
00018 there is nothing for FreeBasic source code yet.
00019
00020 Rather than being a complete tool-chain fb-doc is designed as a bridge
00021 to existing C back-ends, since it's a lot of work to build and test
00022 such a complete tool-chain for several output formats and keep it up to
00023 date. Instead fb-doc creates an intermediate format (similar to the FB
00024 source code, but with C-like syntax) that can be used with existing,
00025 well developed and tested C documenting back-ends. fb-doc has been
00026 tested with
00027 - [gtk-doc](http://developer.gnome.org/gtk-doc-manual/stable/index.html)
00028 - [Doxygen](http://www.doxygen.org/)
00029 The later is used for this documentation.
00030
00031 The steps to generate a well documented project are
00032 -# generate source code, compile and test it
```

```
00033 -# add documentation comments inside the source code and keep them up
00034    to date
00035 -# run fb-doc on the FreeBasic source code to build a C-like
00036    intermediate format
00037 -# run the C-back-end on the intemediate format to build the
00038    documentation output in one or more output formats (ie \em html,
00039    \em pdf, \em manpage, ...)
00040
00041 This intermediate format doesn't contain complete C source and
00042 cannot get compiled by a C compiler. Instead it just contains all
00043 information the lexical scanner of the back-end needs to build the
00044 documentation output. Therefor fb-doc itself doesn't parse the complete
00045 FreeBasic source code. Instead it also acts as a lexical scanner to
00046 extract just the necessary information.
00047
00048 \note That's why fb-doc shouldn't be executed on buggy source (be
00049       prepared to face fancy output in that case).
00050
00051
00052 \section sectIntroFbdoc fb-doc
00053
00054 fb-doc is a multi functional tool, supporting the complete process of
00055 documenting FB source code. You can use it at the command line and
00056 manually control its operations by individual command line options.
00057 It's also designed to operate as an automated tool to act
00058
00059 - in a *Makefile*,
00060 - as a filter for *Doxygen* or
00061 - as a custom command for *Geany*.
00062
00063 Several run modes control where to get input from and where to write
00064 output at. Several emitters are available do generate different kinds
00065 of output. Each run mode has its default emitter, but an individual
00066 emitter can be selected instead. A plugin system makes it possible to
00067 extend fb-doc by external emitters.
00068
00069 All this gets controlled by the command line interface. It may be a bit
00070 overhelming at the beginning. It's recommended to concentrate on the
00071 examples to get startet.
00072
00073
00074 \section sectIntroDocs Documentation Context
00075
00076 The documentation context gets build from the FB source code and its
00077 surrounding comments. fb-doc exports both in a C-like intermediate
00078 format as input for the used back-end.
00079
00080 Not all comments get exported. Only comments starting with a magic
00081 character are used to build the documentation, see \ref sectExaComments
00082 for details.
00083
00084 Also, not all FB source code gets exported. The output is reduced to
00085 the minimal set of constructs to build the documentation, see \ref
00086 sectEmitterCSource for details. This makes the execution of the
00087 tool-chain faster and avoids confusion due to the foreign language for
00088 the lexical scanner.
00089
00090 This constructs and their comments get emitted at the same line number
00091 in the intermediate format to enable the tool-chain referencing to a
00092 certain line.
00093
00094
00095 \section sectIntroSelf About this Text
00096
00097 And finally some words about this documentation (the text you're
00098 currently reading). It is generated by Doxygen back-end with fb-doc
00099 filter. Find examples for a lot of topics by studying the files in the
00100 folder *src* of the fb-doc package. Beside this you may want to check
00101 the files in the folder *doc*. configurations for Doxygen to extract
00102 this pages. A good start may be to use these files and create your
00103 customized fb-doc documentation and play around with the parameters,
00104 see \ref sectExaCli.
00105
00106 To be honest: in some cases this documentation may be a bit
00107 overloaded and serve more information than necessary. But one of the
00108 reasons for creating it is to demonstrate the features of fb-doc and
00109 Doxygen. Therefor not all possibilities are used to reduce the
00110 output to the essentials (see the Doxygen manual for details).
00111
00112 \note It's not under the scope of this documentation to describe the
00113       usage of any (or all possible) tool-chain(s). Please refer to
00114       the respective manual(s) for further information.
```

15.41 src/Options.md File Reference

15.42 src/Options.md

```
00001 Options in Detail  {#pageOptionDetails}
00002 =================
00003 \tableofcontents
00004
00005 This page contains more detailed informations about the options.
00006
00007 fb-doc gets controlled by the command line, containing elements of this
00008 types (may be none, one or more than one)
00009
00010 - options (starting with ''-'' or ''--'')
00011   -# mode options
00012   -# operational options
00013   -# option parameters (separated by white space)
00014 - file specifications (in any combination of)
00015   -# a single file name
00016   -# a list of single file names (separated by white spaces)
00017   -# a file pattern
00018   -# a list of file patterns (separated by white spaces)
00019
00020 \note An empty command line defaults to option `--help`.
00021
00022 The options get evaluated in the given order. When two options are used
00023 with similar effects, fb-doc stops with and error message. Example
00024 (both options are contrary run mode options).
00025
00026 ~~~{.sh}
00027 prompt$ fb-doc --list-mode --geany-mode
00028 fb-doc: Invalid command line (multiple run modes)
00029 ~~~
00030
00031 The options `--help` and `--version` are dominat. They stop further
00032 command line parsing and get executed immediately.
00033
00034 Since a mode option sets its standard emitter (and overrides any
00035 previous emitter settings), it's best practise to set options in the
00036 following order
00037
00038 ~~~{.sh}
00039 fb-doc [mode options] [operational options] <file specifications>
00040 ~~~
00041
00042 \note The options information in the following summary tables belong to
00043       the standard emitters. ???
00044
00045
00046 \section sectOptModes Mode Options
00047
00048 The standard mode is the Doxygen-Filter mode. fb-doc operates in this
00049 mode when no other mode option is set. The only exeption is an empty
00050 command line (no file specification). In this case fb-doc switches to
00051 \ref subsectOptHelp.
00052
00053 \subsection sectoptDefaultMode Default Mode
00054
00055 |       _Summary_ | Standard Mode (Doxygen Filter)                          |
00056 | --------------: | :------------------------------------------------------ |
00057 |           Input | file (path and name)                                    |
00058 |          Output | STDOUT (context depends on emitter)                     |
00059 | Default Emitter | C_Source (in FB style)                                  |
00060 |       File Spec | FreeBasic source code (like *.bas;*.bi -- no default)   |
00061 | Further Options | -a -c -e -r -t (depends on emitter)                     |
00062 | Ignored Options | -o
00063
00064 In standard mode fb-doc reads input from files and sends output to
00065 STDOUT. The default emitter is `C_Source` in default mode. The file
00066 specification may contain a single file name, a path and afile name, a
00067 file pattern or a path and a file pattern. Several file specifications
00068 can be used in any combination, separated by a white space character.
00069 The emitter output for each file gets collected and send in a single
00070 stream to STDOUT.
00071
00072 The standard mode is designed to be used as a filter for Doxygen. In
00073 that case a single file name (including a path, if any) is passed at
00074 the command line. fb-doc operates on this file and emits its output to
00075 STDOUT pipe.
00076
00077 Furthermore this mode is helpful to test the fb-doc output (ie when
00078 developing a new emitter). Just specify your emitter and an input file
00079 like
00080
```

```
00081 ~~~{.sh}
00082 fb-doc --emitter "Plugin" fb-doc.bas
00083 ~~~
00084
00085 and fb-doc operates on the file fb-doc.bas and pipes the output of the
00086 \em Plugin emitter to the terminal.
00087
00088 Or you can use this mode to generate a C-header for a library,
00089 collecting the context of several FB headers in one C header. While it
00090 has some advantages to use several header files during the development,
00091 further users of the library may prefer to have only one file. This can
00092 be done by scanning the source files and piping STDOUT to a file
00093
00094 ~~~{.sh}
00095 fb-doc --emitter C_Source --c_style *.bi > MyLib_SingleHeader.h
00096 ~~~
00097
00098 When your headers need a certain order you may use the option `--tree`
00099 and specify the start files instead of the pattern.
00100
00101 Or you can generate a file for external documentation (outside the
00102 source code). Let fb-doc collect all relevant symbols by using the
00103 `--tree` option as in the example above. Or extract symbols in a per
00104 file basis, like
00105
00106 ~~~{.sh}
00107 fb-doc --emitter "DoxygenTemplates" MyFile_abc.bas > MyFile_abc.txt
00108 ~~~
00109
00110 Then edit all `FIXME` entries in MyFile_abc.txt and include the file in
00111 to your Doxygen file tree.
00112
00113
00114 \subsection subsectOptFileMode File Mode (-f)
00115
00116 |       _Summary_ | File Mode                                        |
00117 | --------------: | :----------------------------------------------- |
00118 |           Input | file (path and name)                             |
00119 |          Output | file (.bi -> .h, .bas -> .c)                     |
00120 | Default Emitter | C_Source (in FB style)                           |
00121 |       File Spec | FreeBasic source code (default: *.bas *.bi)      |
00122 | Further Options | -a -c -e -o (defaults to ../doc/c_src) -r -t     |
00123 | Ignored Options | (depends on emitter)                             |
00124
00125 The file mode is designed to read input from certain files and to write
00126 output to certain files. For each input file an output file gets
00127 created in the output folder specified by option `--outpath`. The type
00128 of the output file depends on the emitter in use:
00129
00130 |            Emitter | `*.bas` Type    | `*.bi` Type    | default path    |
00131 | ------------------: | :-------------- | :------------- | :-------------- |
00132 |           C_Source | --> *.c         | --> *.h        | ../doc/c_src    |
00133 | SyntaxHighLighting | --> *.bas.html  | --> *.bi.html  | ../doc/fb_html  |
00134 |    GtkDocTemplates | --> *.bas.gtmp  | --> *.bi.gtmp  | ../doc/src      |
00135 |   DoxygenTemplates | --> *.bas.dtmp  | --> *.bi.dtmp  | ../doc/src      |
00136 |      FunctionNames | --> *.bas.lfn   | --> *.bi.lfn   | ../doc/src      |
00137 |    external plugin | --> *.bas.ext   | --> *.bi.ext   | ../doc/src      |
00138
00139 fb-doc creates a new folder if the output folder doesn't exist. Also
00140 higher level directories get created if not existend. When an input
00141 file comes from a subfolder, a similar subfolder gets created in the
00142 output folder. When a subfolder is beyond the current path (ie like
00143 `../../../src` form an \#`INCLUDE` statement in a source file) the ..
00144 part(s) of the path get skipped and fb-doc creates subfolders for the
00145 rest of the path.
00146
00147 \note fb-doc never writes above the output folder.
00148
00149 \note Existing files in the output folder get overriden without warning.
00150
00151 fb-doc uses the above mentioned default paths if option `--outpath` is
00152 not used. To write in to the current directory use option `--outpath .`
00153 (path name is a dot).
00154
00155 The most common use is to bridge to the gtk-doc back-end. Usually a set
00156 of FB source files get translated in to similar C source files using
00157 the C_Source emitter. gtk-doc then operates on that C files to generate
00158 the desired output. So usually fb-doc gets executed in the source
00159 folder like
00160
00161 ~~~{.sh}
00162 fb-doc --file-mode --asterix --cstyle --recursiv
00163 ~~~
00164
00165 to operate on all FB files (using default `*.bas` `*.bi`) in the
00166 current folder and its subfolders to generate a similar source tree in
00167 the default output folder <em>../doc/c_src</em>. Then switch to this
```

```
00168 folder and start the gtk-doc tool chain there.
00169
00170 Furthermore you can add a file pattern (or multiple) to operate on
00171 selected files (ie "??_view*.b*"). Or we use this run mode in a
00172 Makefile to update a set of C sources like
00173
00174 ~~~{.sh}
00175 fb-doc --file-mode --asterix --cstyle $@
00176 ~~~
00177
00178 \note The renaming of the output files as in the above table
00179       is inbuild in fb-doc source code and cannot get adapted by
00180       command line settings.
00181
00182
00183 \subsection subsectOptGeanyMode Geany Mode (-g)
00184
00185 |       _Summary_ |  Mode                                |
00186 | --------------: | :----------------------------------- |
00187 |           Input | STDIN                                |
00188 |          Output | STDOUT                               |
00189 | Default Emitter | GtkDocTemplates                      |
00190 |       File Spec | none                                 |
00191 | Further Options | -e                                   |
00192 | Ignored Options | -a -c -o -r -t (depends on emitter)  |
00193
00194 The geany mode is designed to generate templates for the
00195 documentational comments in the source code, see section \ref
00196 sectInstallGeany for details. Usually a code section gets selected in
00197 the editor and then sent to fb-doc via STDIN pipe. fb-doc extracts the
00198 relevant symbols and generates a matching comment block for this piece
00199 of code. Both, the comment block and the original code block, get
00200 returned to geany and replaces the previously selected block.
00201
00202 From a geany user point of view it looks like adding a block in to the
00203 source, but this block contains individual informations related to the
00204 previously selected block.
00205
00206 Geany mode is also usefull for testing purposes because the output to
00207 STDOUT doesn't change any files in your source tree. Just edit the
00208 custom command in Geany settings to change to the desired emitter. This
00209 option allows to specify an emitter name directly after the option
00210 (since geany mode doesn't need any file specifications) but you can
00211 also use option `--emitter` in this mode. The output can either be
00212 tested in the geany editor, where you can easy select the stuff for
00213 fb-doc. Or you can pipe an input stream to fb-doc at the command line.
00214 The following example lists all function names from file test.bas in
00215 the terminal
00216
00217 ~~~{.sh}
00218 fb-doc --geany "FunctionNames" < test.bas
00219 ~~~
00220
00221 \note The output may contain error messages from fb-doc. Those are send
00222       to STDERR and mixed in to the output by the terminal. That's
00223       different when directing the STDOUT stream to a file. In that
00224       case the error messages are shown in the terminal while the file
00225       contains pure fb-doc output.
00226
00227 Furthermore you can use this mode to collect output from several files
00228 in to a single one. The following example collects the C-translations of
00229 the declarations from two FB headers in the file C_Header.h
00230
00231 ~~~{.sh}
00232 fb-doc --geany "C_Source" --c-style < file1.bi > C_Header.h
00233 fb-doc --geany "C_Source" --c-style < file2.bi >> C_Header.h
00234 ~~~
00235
00236
00237 \subsection subsectOptListMode List Mode (-l)
00238
00239 |       _Summary_ |  Mode                                          |
00240 | --------------: | :--------------------------------------------- |
00241 |           Input | file (path from Doxyfile, name = *.bas;*.bi)   |
00242 |          Output | fb-doc.lfn                                     |
00243 | Default Emitter | ListOfFunction                                 |
00244 |       File Spec | Doxyfile (but also *.bas;*.bi)                 |
00245 | Further Options | -a -c -e -r -t                                 |
00246 | Ignored Options | -o (depends on emitter)                        |
00247
00248 The list mode is designed to generate a list of function names and
00249 write this list to the file fb-doc.lfn, one function name per line.
00250 Later, this file is used to generate pseudo function calls in the
00251 function bodies by the C_Source emitter, making Doxygen being able to
00252 generate caller / callee graphs.
00253
00254 Usually the mode is used in the `../doc` folder near the Doxyfile.
```

```
00255 fb-doc reads the 'INPUT' path from the Doxyfile and scans all FB
00256 source files, to extract the function names in a single file
00257 'fb-doc.lfn', which gets written near the Doxyfile (where fb-doc can
00258 find it later on when called by Doxygen as input filter). Example
00259
00260 ~~~{.sh}
00261 cd ../doc
00262 fb-doc --list-mode
00263 ~~~
00264
00265 In this example no input file name is specified and fb-doc uses the
00266 default name 'Doxyfile'. Also it's possible to specify any outher file
00267 name and a prepending path (ie like '../doc/fb-doc.Doxyfile'). But if
00268 the file specification contains a pattern (characters '*' or '?') or
00269 the extension is one of '.bas' or '.bi', then fb-doc skips the reading
00270 of the Doxyfile and operates on the FB source file(s) directly.
00271
00272
00273 \subsection subsectOptSyntaxMode Syntax Mode (-s)
00274
00275 |       _Summary_ | Syntax Mode                                |
00276 | --------------: | :----------------------------------------- |
00277 |           Input | files (pathes and names from Doxyfile)     |
00278 |          Output | files                                      |
00279 | Default Emitter | SyntaxHighLighting                         |
00280 |       File Spec | Doxyfile                                   |
00281 | Further Options | -e -o                                      |
00282 | Ignored Options | -a -c -r -t (depends on emitter)           |
00283
00284 In syntax mode fb-doc opens the specified Doxyfile and evaluates the
00285 path to the FB source and the output types and their folders. Then
00286 it scans the output folders for files containing listings, operating
00287 on Html, LaTeX and XML output.
00288
00289 These listing files get replaced by a version with the original
00290 header and footer and inbetween a newly created listing section with
00291 appropriate FB syntax highlighting. All links found in the original
00292 files get transfered to the new file.
00293
00294 The syntax mode is designed to correct syntax highlighting in the
00295 Doxygen output files. Since the paths get evaluated from the
00296 Doxyfile, no options (additional to '--syntax-mode') are neccessary.
00297 By default fb-doc read the file named "Doxyfile", but its possible
00298 to specify (one or more) individual file name(s).
00299
00300 Since this mode operates on specific Doxygen output files, it makes no
00301 sense to use it with an alternative emitter (but fb-doc doesn't block
00302 this scenario).
00303
00304
00305 \subsection subsectOptHelp Help (-h)
00306
00307 |       _Summary_ | Help Mode              |
00308 | --------------: | :--------------------- |
00309 |           Input | none                   |
00310 |          Output | STDOUT (version text)  |
00311 | Default Emitter | none                   |
00312 |       File Spec | none                   |
00313 | Further Options | none                   |
00314 | Ignored Options | none                   |
00315
00316 This option makes fb-doc to output the help text and stop. The help
00317 text contains a brief summary of all available option and some examples
00318 for usage. It should help experienced users to remember some
00319 information. (It's not mentioned to be a complete documentation.)
00320
00321 \note In help mode all other options have no effect.
00322
00323
00324 \subsection subsectOptVersion Version (-v)
00325
00326 |       _Summary_ | Version Mode          |
00327 | --------------: | :-------------------- |
00328 |           Input | none                  |
00329 |          Output | STDOUT (help text)    |
00330 | Default Emitter | none                  |
00331 |       File Spec | none                  |
00332 | Further Options | none                  |
00333 | Ignored Options | none                  |
00334
00335 This option makes fb-doc to output the version information and stop.
00336 The version information contains the source code version number, the
00337 date and time of compilation and the used operating system for the fbc
00338 compiler.
00339
00340 \note In version mode all other options have no effect.
00341
```

```
00342
00343 \section sectOptOperaOptions Operational Options
00344
00345 Operational options control the fb-doc operations or the output of an
00346 emitter. In special cases they may have no effect, depending on the
00347 used combination of run mode and emitter.
00348
00349 \subsection subsectOptAsterix Asterix (-a)
00350
00351 |            `-a` | `--asterix`          |
00352 | --------------: | :------------------- |
00353 |       Parameter | none                 |
00354 |        Run Modi | all                  |
00355 |        Emitters | `C_Source`           |
00356
00357 This options makes the emitter `C_Source` to output an asterix
00358 character and a white space (like `"* "`) at the start of each line in
00359 a special multi line comment block, see \ref subsectExaGtkdoc for
00360 further examples.
00361
00362 This special format is used in gtk-doc (must have) and can also be used
00363 in Doxygen (no advantage, slows down execution).
00364
00365 Editing and formating a special comment block with these line starts is
00366 complicated and slow. The asterix character max get mixed up with the
00367 text when using line wrapping functions. Therefor fb-doc offers this
00368 feature to edit the documentation context in plain text and add the
00369 special format only in the output for the back-end.
00370
00371
00372 \subsection subsectOptCstyle CStyle (-c)
00373
00374 |            `-c` | `--cstyle`                    |
00375 | --------------: | :---------------------------- |
00376 |       Parameter | none                          |
00377 |        Run Modi | all                           |
00378 |        Emitters | `C_Source`, `DoxygenTemplates` |
00379
00380 This option makes fb-doc to emit real C types instead of the FB-like
00381 mangled type names. It also infuences the translation of `TYPE` blocks
00382 and \#`INCLUDE` lines, see \ref sectTabInterForm for examples.
00383
00384 The standard output of the `C_Source` emitter is optimized for best
00385 matching documentation. Therefor the types in the source code get
00386 fantasy names, similar to the FB keywords declaring them. Use this
00387 option to switch form FB sytle to real C type names.
00388
00389
00390 \subsection subsectOptEmitters Emitter (-e)
00391
00392 |            `-e` | `--emitter`          |
00393 | --------------: | :------------------- |
00394 |       Parameter | Emitter name         |
00395 |        Run Modi | all                  |
00396
00397 This option makes fb-doc to use an alternative emitter and overrides
00398 the run mode default emitter setting. A parameter must follow this
00399 option (separated by a white space), specifying the emitter name. The
00400 parameter may be surrounded by quotes (single or double), they get
00401 removed befor further operation.
00402
00403 First, fb-doc searches in the list of internal emitter names (see \ref
00404 sectTabEmitters). This search gets done non-case-sensitive and
00405 partial-matching. Meaning you need not type the complete emitter name
00406 nor use the right letter cases. Ie *d*, *Dox* or <em>"DOXY"</em> all
00407 match the full emitter name *DoxygenTemplates*.
00408
00409 In case of no match in the internal emitter names fb-doc tries to load
00410 an external emitter with the specified name. In this case the emitter
00411 name must exactly match the base file name of the FB source code used
00412 to build the plugin module. Ie when the plugin was compiled by
00413
00414 ~~~{.sh}
00415 fbc -dylib Plugin.bas
00416 ~~~
00417
00418 the parameter *emitter name* must be `Plugin` as in
00419
00420 ~~~{.sh}
00421 fbdoc -e "Plugin"
00422 ~~~
00423
00424 \note On UNIX-like systems file names are case-sensitive.
00425 \note For external emitter plugins specify the base name of the source
00426       file (not the context of the string \ref EmitterIF::Nam).
00427
00428
```

```
00429 \subsection subsectOptOutpath Outpath (-o)
00430
00431 |            `-o` | `--outpath`                 |
00432 | --------------: | :-------------------------- |
00433 |       Parameter | path to folder              |
00434 |        Run Modi | `--file-mode`, `--list-mode` |
00435 |        Emitters | all                         |
00436
00437 This option is used to specify the path for the fb-doc file output. It
00438 works for the above mentioned run modi, wherein fb-doc generates new
00439 file output (but not for `--syntax-mode` where fb-doc replaces files
00440 generated by Doxygen).
00441
00442 The parameter may be either a relative path starting at the current
00443 directory (where fb-doc is executed) or an absolute path (starting with
00444 "/" on UNIX-like systems or with a drive letter and a colon an other
00445 systems).
00446
00447 fb-doc writes the file output in the specified directory. The directory
00448 gets created first, if it doesn't exist. Also all higher level
00449 directories get created if not existing yet.
00450
00451 In `--list-mode` the file *fb-doc.lfn* gets created in that directory,
00452 overriding an existed without warning (if any).
00453
00454 In `--file-mode` more than one file may get created, depending on the
00455 specified input file(s) or pattern(s). In case of option `--recursiv`
00456 or `--tree` also subdirectories may get created in the putpath
00457 directory and its subfolder(s).
00458
00459 \note fb-doc writes files in to the outpath folder and may create
00460       subfolders in it. But fb-doc never performs any changes in
00461       directories above the outpath.
00462
00463
00464 \subsection subsectOptRecursiv Recursiv (-r)
00465
00466 |            `-r` | `--recursiv`          |
00467 | --------------: | :-------------------- |
00468 |       Parameter | none                  |
00469 |        Run Modi | all                   |
00470 |        Emitters | all                   |
00471
00472 This options makes fb-doc scanning for input files in the working
00473 folder and in its subfolders. It takes only into effect when the input
00474 file specification is a file pattern (or a list of patterns).
00475
00476 When the file pattern has no path, the current folder is the working
00477 folder. Otherwise fb-doc scans the path specified before the pattern
00478 and its subfolders.
00479
00480 In the following example the current folder is *doc* and fb-doc scans
00481 the working folder *src* and its subfolders
00482
00483 ~~~{.sh}
00484 cd myProj/doc
00485 fbdoc -r "../src/*.bas" "../src/*.bi"
00486 ~~~
00487
00488 \note The option has no effect when a single file name (or a list of
00489       names) is specified (it's only used for patterns).
00490 \note When fb-doc operates on a Doxyfiles (*in* `--list-mode` *or*
00491       `--syntax-mode`) the setting of its *RECURSIV* parameter
00492       overrides this option.
00493 \note It has no effect when running in `--geany-mode`.
00494 \note On LINUX systems usually the shell (bash) expands the file
00495       patterns and sends a list of single names to fb-doc, so this
00496       option has no effect until you enclose the file pattern by
00497       quotation marks (like in the example above).
00498
00499
00500 \subsection subsectOptTree Tree (-t)
00501
00502 |            `-t` | `--tree`              |
00503 | --------------: | :-------------------- |
00504 |       Parameter | none                  |
00505 |        Run Modi | all                   |
00506 |        Emitters | all                   |
00507
00508 This option makes fb-doc to follow the source code tree. That is, all
00509 \#`INCLUDE` statements will be evaluated and fb-doc operates on this
00510 files as if they were specified as input files on the command line.
00511
00512 \note This only works with files in the source code tree. Standard
00513       header files (ie like `"crt/string.bi"`) wont be found since
00514       fb-doc doesn't know the standard FreeBasic include path.
00515 \note For this option to work the emitter must provide a handler for
```

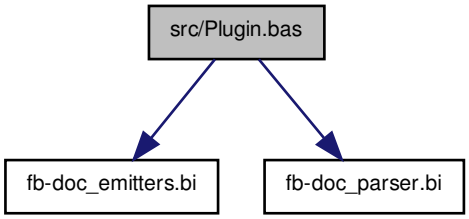
```
00516       \ref EmitterIF::Incl_() in which the parsing of the new files
00517       get started. Not all emitters do support this.
00518
00519
00520 \section sectOptFileSpec File Specifications
00521
00522 A file specification is used to determine one or more file(s) for
00523 fb-doc file input (so not for `--geany-mode` where fb-doc gets input
00524 from STDIN). Each entry at the command line that is neither an option
00525 nor an option parameter gets recognized as file specification and added
00526 to the list \ref Options::InFiles for further operation.
00527
00528 A file specification contains an (optional) path to the working
00529 directory and either a concrete file name or a file pattern. Examples
00530
00531 | File Specification | Description                                                          |
00532 | -----------------: | :------------------------------------------------------------------- |
00533 |          `abc.bas` | the file `abc.bas` in the current directory                          |
00534 |  `../doc/Doxyfile` | the file named `Doxyfile` in the working folder `../doc`             |
00535 |            `*.bas` | all files matching the pattern `*.bas` in the current folder         |
00536 |      `../src/*.bi` | all files matching the pattern `*.bi` in the working folder `../src` |
00537
00538 Whereat the current directory is the folder fb-doc was executed in. And
00539 the working folder is the directory specified by the path part of the
00540 file specification. The later can either be a relative path (as in the
00541 examples above) or an absolute path (starting with "/" on UNIX-like
00542 systems or with a drive letter and a colon on other systems).
00543
00544 File specifications can be used in any combinations. Use white spaces to
00545 separate them. When a path or a file name contains a white space it
00546 must be enclosed by (single or double) quotes. Generally it's
00547 benefiting to enclose any file specification in qoutes, especially on
00548 UNIX-like systems whereat unquoted pattern get expanded in the shell
00549 and fb-doc receives a list of names instead of the pattern (option
00550 `--recursiv` doesn't work in that case).
00551
00552 fb-doc uses default file specifications if none is set in the command
00553 line. The default depends on the specified run mode:
00554
00555 |        Run Mode | Default File Specification |
00556 | --------------: | :------------------------- |
00557 |    default mode | `*.bas` `*.bi`             |
00558 |   `--file-mode` | `*.bas` `*.bi`             |
00559 |   `--list-mode` | `Doxyfile`                 |
00560 | `--syntax-mode` | `Doxyfile`                 |
```

15.43 src/Plugin.bas File Reference

An example 'Code for an external emitter.

```
#include "fb-doc_emitters.bi"
#include "fb-doc_parser.bi"
```

Include dependency graph for Plugin.bas:



Functions

- SUB_CDECL dll_declare (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at a variable declaration.
- SUB_CDECL dll_function (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is on top of a function body.
- SUB_CDECL dll_enum (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at the start of a ENUM block.
- SUB_CDECL dll_union (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at the start of a UNION block.
- SUB_CDECL dll_class (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at the start of a TYPE block.
- SUB_CDECL dll_define (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at an #DEFINE line or at the start of a #MACRO
- SUB_CDECL dll_include (BYVAL_AS_Parser_PTR P)

  Emitter called when the Parser is at an #INCLUDE line.
- SUB_CDECL dll_init (BYVAL_AS_Parser_PTR P)

  Emitter called before the input gets parsed.
- SUB_CDECL dll_error (BYVAL_AS_Parser_PTR P)

  Emitter called for an error.
- SUB_CDECL dll_empty (BYVAL_AS_Parser_PTR P)

  Emitter called for an empty block in mode --geany-mode
- SUB_CDECL dll_exit (BYVAL_AS_Parser_PTR P)

  Emitter called after the input got parsed.
- SUB_CDECL dll_CTOR (BYVAL_AS_Parser_PTR P)

  Emitter called after the input got parsed.
- SUB_CDECL dll_DTOR (BYVAL_AS_Parser_PTR P)

  Emitter called after the input got parsed.
- FUNCTION_CDECL_AS_EmitterIF_PTR EmitterInit ()

  Function called by fb-doc to get the EmitterIF.

15.43.1 Detailed Description

An example 'Code for an external emitter. This file contains example source 'Code for an external emitter. It isn't used in the fb-doc source tree. See Extending fb-doc for details.

Definition in file Plugin.bas.

15.43.2 Function Documentation

15.43.2.1 SUB_CDECL dll_declare (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at a variable declaration.

This emitter generates a list of the function names called via the emitter interface. So when you input some source to fb-doc and use this emitter, the output is a list of the functions called by the parser for this input.

Before you can use this emitter, you have to compile it first, using the command

```
fbc -dylib Plugin.bas
```

The result is a binary called

- libdll_emitter.so (LINUX)

- libdll_emitter.dll (windows)

There's no way to compile or use an external emitter on DOS since DOS doesn't support dynamic linked libraries.

To use this emitter in fb-doc set its name (without the suffix .bas) just as an internal name (,so don't name your customized emitter similar to an internal emitter name).

Ie. the emitter output for the context of this file can be viewed in the terminal by

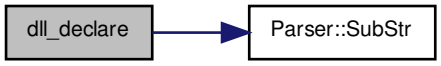
```
./fb-doc --emitter "Plugin" Plugin.bas
```

(LINUX example) and will generate the following output:

```
DLL_INIT
DLL_INCLUDE
DLL_INCLUDE
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_FUNCTION
DLL_EXIT
```

Definition at line 67 of file Plugin.bas.

Here is the call graph for this function:

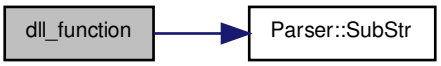


15.43.2.2 SUB_CDECL dll_function (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is on top of a function body.

Definition at line 72 of file Plugin.bas.

Here is the call graph for this function:



15.43.2.3 SUB_CDECL dll_enum (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at the start of a ENUM block.

Definition at line 80 of file Plugin.bas.

15.43.2.4 SUB_CDECL dll_union (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at the start of a UNION block.

Definition at line 85 of file Plugin.bas.

15.43.2.5 SUB_CDECL dll_class (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at the start of a TYPE block.

Definition at line 90 of file Plugin.bas.

15.43.2.6 SUB_CDECL dll_define (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at an #DEFINE line or at the start of a #MACRO

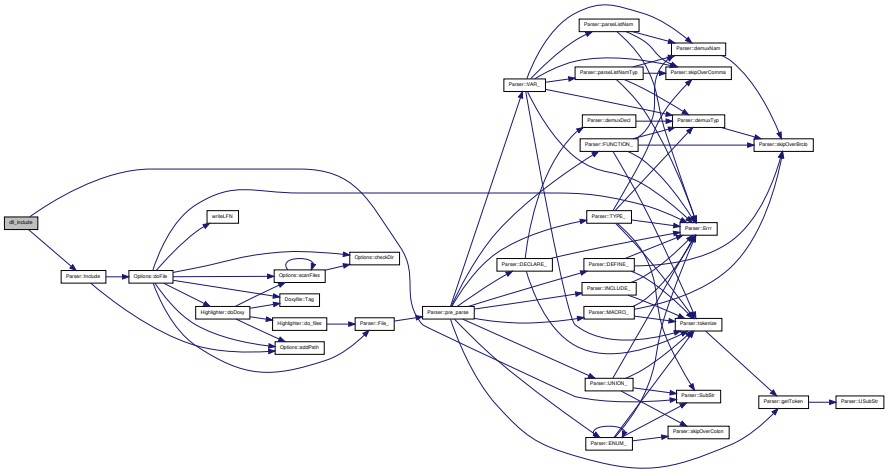
Definition at line 95 of file Plugin.bas.

15.43.2.7 SUB_CDECL dll_include (BYVAL_AS_Parser_PTR *P*)

Emitter called when the Parser is at an #INCLUDE line.

Definition at line 100 of file Plugin.bas.

Here is the call graph for this function:



15.43.2.8 SUB_CDECL dll_init (BYVAL_AS_Parser_PTR *P*)

Emitter called before the input gets parsed.

Definition at line 109 of file Plugin.bas.

15.43.2.9 SUB_CDECL dll_error (BYVAL_AS_Parser_PTR *P*)

Emitter called for an error.

Definition at line 114 of file Plugin.bas.

15.43.2.10 SUB_CDECL dll_empty (BYVAL_AS_Parser_PTR *P*)

Emitter called for an empty block in mode `--geany-mode`

Definition at line 119 of file Plugin.bas.

15.43.2.11 SUB_CDECL dll_exit (BYVAL_AS_Parser_PTR *P*)

Emitter called after the input got parsed.

Definition at line 124 of file Plugin.bas.

15.43.2.12 SUB_CDECL dll_CTOR (BYVAL_AS_Parser_PTR *P*)

Emitter called after the input got parsed.

Definition at line 129 of file Plugin.bas.

15.43.2.13 SUB_CDECL dll_DTOR (BYVAL_AS_Parser_PTR *P*)

Emitter called after the input got parsed.

Definition at line 134 of file Plugin.bas.

15.43.2.14 FUNCTION_CDECL_AS_EmitterIF_PTR EmitterInit ( )

Function called by fb-doc to get the EmitterIF.

Definition at line 158 of file Plugin.bas.

15.44 Plugin.bas

```
%%% Syntax-highlighting by fb-doc %%%
00001 /'*
00002 \file Plugin.bas
00003 \brief An example 'Code for an external emitter
00004
00005 This file contains example source 'Code for an external emitter. It
00006 isn't used in the fb-doc source tree. See \ref pageExtend for details.
00007
00008 '/
00009
00010 /'*
00011
00012 This emitter generates a list of the function names called via the
00013 emitter interface. So when you input some source to fb-doc and use
00014 this emitter, the output is a list of the functions called by the
00015 parser for this input.
00016
00017 Before you can use this emitter, you have to compile it first, using
00018 the command
00019
00020 \code fbc -dylib Plugin.bas \endcode
00021
00022 The result is a binary called
00023
00024   - libdll_emitter.so  (LINUX)
00025   - libdll_emitter.dll (windows)
00026
```

```
00027 There's no way to compile or use an external emitter on DOS since
00028 DOS doesn't support dynamic linked libraries.
00029
00030 To use this emitter in fb-doc set its name (without the suffix .bas)
00031 just as an internal name (,so don't name your customized emitter
00032 similar to an internal emitter name).
00033
00034 Ie. the emitter output for the context of this file can be viewed in
00035 the terminal by
00036 ~~~{.sh}
00037 ./fb-doc --emitter "Plugin" Plugin.bas
00038 ~~~
00039 (LINUX example) and will generate the following output:
00040
00041 \verbatim
00042 DLL_INIT
00043 DLL_INCLUDE
00044 DLL_INCLUDE
00045 DLL_FUNCTION
00046 DLL_FUNCTION
00047 DLL_FUNCTION
00048 DLL_FUNCTION
00049 DLL_FUNCTION
00050 DLL_FUNCTION
00051 DLL_FUNCTION
00052 DLL_FUNCTION
00053 DLL_FUNCTION
00054 DLL_FUNCTION
00055 DLL_FUNCTION
00056 DLL_FUNCTION
00057 DLL_EXIT
00058 \endverbatim
00059
00060 '/
00061
00062
00063 #INCLUDE ONCE "fb-doc_emitters.bi" ' declaration of the emitter interface
00064 #INCLUDE ONCE "fb-doc_parser.bi"   ' declaration of the Parser members (not used here)
00065
00066 '* \brief Emitter called when the Parser is at a variable declaration
00067 SUB dll_declare CDECL(BYVAL P AS Parser PTR)
00068   Code(NL & __FUNCTION__ & ": " & P->LineNo & " " & P->SubStr(P->NamTok))
00069 END SUB
00070
00071 '* \brief Emitter called when the Parser is on top of a function body
00072 SUB dll_function CDECL(BYVAL P AS Parser PTR)
00073   WITH *P
00074     VAR nam = .SubStr(IIF(.NamTok[3] = .TOK_DOT, .NamTok[6], .NamTok))
00075     Code(NL & __FUNCTION__ & ": " & .LineNo & " " & nam)
00076   END WITH
00077 END SUB
00078
00079 '* \brief Emitter called when the Parser is at the start of a ENUM block
00080 SUB dll_enum CDECL(BYVAL P AS Parser PTR)
00081   Code(NL & __FUNCTION__)
00082 END SUB
00083
00084 '* \brief Emitter called when the Parser is at the start of a UNION block
00085 SUB dll_union CDECL(BYVAL P AS Parser PTR)
00086   Code(NL & __FUNCTION__)
00087 END SUB
00088
00089 '* \brief Emitter called when the Parser is at the start of a TYPE block
00090 SUB dll_class CDECL(BYVAL P AS Parser PTR)
00091   Code(NL & __FUNCTION__)
00092 END SUB
00093
00094 '* \brief Emitter called when the Parser is at an \#`DEFINE` line or at the start of a \#`MACRO`
00095 SUB dll_define CDECL(BYVAL P AS Parser PTR)
00096   Code(NL & __FUNCTION__)
00097 END SUB
00098
00099 '* \brief Emitter called when the Parser is at an \#`INCLUDE` line
00100 SUB dll_include CDECL(BYVAL P AS Parser PTR)
00101   WITH *P
00102     VAR nam = .SubStr(.NamTok)
00103     Code(NL & __FUNCTION__ & ": " & .LineNo & " " & nam)
00104     IF .InTree THEN .Include(TRIM(nam, """))
00105   END WITH
00106 END SUB
00107
00108 '* \brief Emitter called before the input gets parsed
00109 SUB dll_init CDECL(BYVAL P AS Parser PTR)
00110   Code(NL & __FUNCTION__)
00111 END SUB
00112
00113 '* \brief Emitter called for an error
```

```
00114 SUB dll_error CDECL(BYVAL P AS Parser PTR)
00115   Code(NL & __FUNCTION__)
00116 END SUB
00117
00118 '* \brief Emitter called for an empty block in mode `--geany-mode`
00119 SUB dll_empty CDECL(BYVAL P AS Parser PTR)
00120   Code(NL & __FUNCTION__)
00121 END SUB
00122
00123 '* \brief Emitter called after the input got parsed
00124 SUB dll_exit CDECL(BYVAL P AS Parser PTR)
00125   Code(NL & __FUNCTION__ & NL)
00126 END SUB
00127
00128 '* \brief Emitter called after the input got parsed
00129 SUB dll_CTOR CDECL(BYVAL P AS Parser PTR)
00130   PRINT __FUNCTION__
00131 END SUB
00132
00133 '* \brief Emitter called after the input got parsed
00134 SUB dll_DTOR CDECL(BYVAL P AS Parser PTR)
00135   PRINT __FUNCTION__
00136 END SUB
00137
00138
00139 ' Setting the EmitterIF pointer (delete un-needed rows here, the string literal is optional)
00140 WITH_NEW_EMITTER("Plugin")
00141     .Decl_ = @dll_declare
00142     .Func_ = @dll_function
00143     .Enum_ = @dll_enum
00144     .Unio_ = @dll_union
00145     .Clas_ = @dll_class
00146     .Defi_ = @dll_define
00147     .Incl_ = @dll_include
00148     .Init_ = @dll_init
00149    .Error_ = @dll_error
00150    .Empty_ = @dll_empty
00151     .Exit_ = @dll_exit
00152     .CTOR_ = @dll_CTOR
00153     .DTOR_ = @dll_DTOR
00154 END WITH
00155
00156
00157 '* \brief Function called by fb-doc to get the \ref EmitterIF
00158 FUNCTION EmitterInit CDECL() AS EmitterIF PTR EXPORT
00159   PRINT __FUNCTION__
00160   RETURN Emitters(0)
00161 END FUNCTION
00162
00163
```

15.45 src/Tables.md File Reference

15.46 src/Tables.md

```
00001 Tables  {#pageTables}
00002 ======
00003 \tableofcontents
00004
00005 This page contains some information on fb-doc in table format.
00006
00007 \section sectTabOverview Overview
00008
00009 fb-doc is a multi functional tool. While several \ref sectTabRunModi
00010 specify where to get input from and where to write output to, several
00011 \ref sectTabEmitters generate different kind of output. Here's a table
00012 of all run modes against the inbuild emitters ("DEF" = default
00013 configuration, "+" = useful combination, "-" = combination not useful).
00014
00015 | Emitter \ Run Mode | default | `--file-mode` | `--list-mode` | `--syntax-mode` | `--geany-mode` |
00016 | -----------------: | :-----: | :-----------: | :-----------: | :-------------: | :------------: |
00017 | C_Source           |   DEF   |      DEF      |       -       |        -        |       +        |
00018 | GtkDocTemplates    |    +    |       +       |       -       |        -        |      DEF       |
00019 | DoxygenTemplates   |    +    |       +       |       -       |        -        |       +        |
00020 | FunctionNames      |    +    |       +       |      DEF      |        -        |       +        |
00021 | SyntaxHighLighting |    +    |       +       |       -       |       DEF       |       +        |
00022
00023
00024 \section sectTabOptions Options
00025
00026 The following table contains an overview of all fb-doc options. An
```

```
00027 option either starts by a minus character followed by a single
00028 character (short form) or by two minus characters followed by a word or
00029 pair of words (long form). Both forms have the same meaning.
00030
00031 Some options expect an additional parameter. Each further word in the
00032 command line (without the leading '-' character) gets interpreted as a
00033 file name or pattern, see \ref pageOptionDetails for details.
00034
00035 <table>
00036 <tr>
00037 <th> Option <th> Parameter <th> Description
00038 </th>
00039 <tr>
00040 <td> \ref subsectOptAsterix
00041 <td> none
00042 <td> The C_Source emitter generates lines in a multi line comment block
00043      with leading '* ' characters (gtk-doc style).
00044 <tr>
00045 <td> \ref subsectOptCstyle
00046 <td> none
00047 <td> The C_Source emitter generates types in real C syntax
00048      (instead of FB styled pseudo C syntax). Also used in emitter
00049      `DoxygenTemplates`.
00050 <tr>
00051 <td> \ref subsectOptEmitters
00052 <td> Emitter name
00053 <td> Customized emitter selection. fb-doc compares the parameter with
00054      the names of the internal emitters. In case of no match it tries to
00055      find and load an external with this name.
00056 <tr>
00057 <td> \ref subsectOptFileMode
00058 <td> none
00059 <td> Read FB source files and write output to similar named files
00060      (overriding existend, if any).
00061 <tr>
00062 <td> \ref subsectOptGeanyMode
00063 <td> Emitter name
00064 <td> Read input from STDIN and write to STDOUT. The parameter is optional.
00065 <tr>
00066 <td> \ref subsectOptHelp
00067 <td> none
00068 <td> Print the help text and stop.
00069 <tr>
00070 <td> \ref subsectOptListMode
00071 <td> none
00072 <td> Read FB source files and write output to file *fb-doc.lfn*.
00073 <tr>
00074 <td> \ref subsectOptOutpath
00075 <td> path
00076 <td> Set folder for file output.
00077 <tr>
00078 <td> \ref subsectOptRecursiv
00079 <td> none
00080 <td> Scan for input files in the working folder and all subfolders.
00081      Works only in combination with file patterns.
00082 <tr>
00083 <td> \ref subsectOptSyntaxMode
00084 <td> none
00085 <td> Scan a Doxyfile (or multiple) for its output and fix syntax
00086      highlighting.
00087 <tr>
00088 <td> \ref subsectOptTree
00089 <td> none
00090 <td> Follow the \#`INCLUDE` statements in the source tree (if
00091      possible -- not available in \ref subsectOptGeanyMode).
00092 <tr>
00093 <td> \ref subsectOptVersion
00094 <td> none
00095 <td> Print the version information and stop.
00096 </table>
00097
00098 \note An empty command line makes fb-doc to output the help text
00099       (as if option `--help` was given).
00100
00101
00102 \section sectTabEmitters Inbuild Emitters
00103
00104 fb-doc has five inbuild emitters to generate different kinds of output.
00105 All \ref sectOptModes set their default emitter, but you can customize
00106 selection by option `--emitter` (`-e`) afterwards. Also this option is
00107 used to choose an external emitter (plugin), see \ref pageExtend for
00108 details.
00109
00110 \note Only one emitter can be in use at a time.
00111
00112 <table>
00113 <tr>
```

```
00114 <th> Name
00115 <th> Default in Mode
00116 <th> Function
00117 </th>
00118 <tr>
00119 <td> \ref sectEmitterCSource
00120 <td> `--file-mode` and default (Doxy-Filter)
00121 <td> Translated FB source in intermediate format (C-like syntax). Use
00122      FB-like typenames (default) or real C syntax (option `--cstyle`).
00123      Prepend asterix in comment blocks (option `--asterix`).
00124 <tr>
00125 <td> \ref sectEmitterGtkTempl
00126 <td> `--geany-mode`
00127 <td> Emit source code and prepend documentation relevant constructs by
00128      templates for gtk-doc. Prefered usage in `--geany-mode`.
00129 <tr>
00130 <td> \ref sectEmitterDoxyTempl
00131 <td> none
00132 <td> Emit source code and prepend documentation relevant constructs by
00133      templates for Doxygen. Prefered usage in `--geany-mode`.
00134 <tr>
00135 <td> \ref sectEmitterLfn
00136 <td> `--list-mode`
00137 <td> Emit a list of all function names. Prefered usage in `--list-mode`
00138      to generate the file *fb-doc.lfn*.
00139 <tr>
00140 <td> \ref sectEmitterSyntax
00141 <td> `--syntax-mode`
00142 <td> Emit all source code surrounded by syntax highlighting tags. In
00143      `--syntax-mode` for Doxygen output in <em>*.html, *.tex</em> and
00144      <em>*.xml</em> files. In other modes *html* tags only.
00145 </table>
00146
00147
00148 \section sectTabRunModi Input / Output
00149
00150 Some options control the execution of fb-doc and the streams of input
00151 and output data. The following table shows the relations between run
00152 mode and input / output targets, See \ref sectOptModes for details.
00153
00154 <table>
00155 <tr>
00156 <th> Option
00157 <th> Input
00158 <th> Output
00159 <th> Default Emitter
00160 </th>
00161 <tr>
00162 <td> none (Doxy-Filter-Mode)
00163 <td> one file
00164 <td> standard out (STDOUT)
00165 <td> C_Source
00166 <tr>
00167 <td> `-f` *or* `--file-mode`
00168 <td> one file, a list of file names, a file pattern, a list of file patterns or all source files
00169 <td> *.\em c and *.\em h files
00170 <td> C_Source
00171 <tr>
00172 <td> `-g` *or* `--geany-mode`
00173 <td> standard in (STDIN)
00174 <td> standard out (STDOUT)
00175 <td> GtkDocTemplates
00176 <tr>
00177 <td> `-l` *or* `--list-mode`
00178 <td> files matching a pattern or all source files
00179 <td> file \em fb-doc.lfn
00180 <td> FunctionNames
00181 <tr>
00182 <td> `-h` *or* `--help`
00183 <td> none
00184 <td> standard out (STDOUT)
00185 <td> none
00186 <tr>
00187 <td> `-v` *or* `--version`
00188 <td> none
00189 <td> standard out (STDOUT)
00190 <td> none
00191 </table>
00192
00193 \note the default mode is the Doxygen filter mode, since Doxygen
00194       doesn't allow to send further options (directly).
00195
00196 \note Input and output channels are fixed to the listed options.
00197       But you can apply changes, ie by specifying an other emitter
00198       with option `--emitter` or input more than one file by walking
00199       through the source tree with option `--tree`, ...
00200
```

```
00201 \note The modi `--help` (`-h`) and `--version` (`-v`)
00202       are not related to any FB source. Instead of using an emitter
00203       they create fixed output on standard output channel (STDOUT).
00204
00205 \note When more then one run mode is specified in the command line,
00206       the latest is the dominant.
00207
00208
00209 \section sectTabFileNames File Specifications
00210
00211 Any text in the command line not matching an option or its parameter
00212 gets interpreted as a file name or file pattern. They get added to a
00213 queue and fb-doc executes this queue in the given order. Depending on
00214 the file name and the run mode fb-doc operates in different ways:
00215
00216 <table>
00217 <tr>
00218 <th> Option (run-mode)
00219 <th> no name
00220 <th> exact name
00221 <th> pattern
00222 </th>
00223 <tr>
00224 <td> default
00225 <td> print help text
00226 <td> load file, emit output to STDOUT
00227 <td> load all files matching the pattern, emit output to STDOUT
00228 <tr>
00229 <td> -f *or* --file-mode <td> load all *.\em bas and *.\em
00230     bi files, emit output for each file to a *.\em c or *.\em h file
00231
00232 <td> load this one file, scan the source tree and load each
00233     \#`INCLUDE` file, emit output for each file to a *.\em c or *.\em
00234     h file
00235 <td> load all files matching the pattern, emit output for each file
00236     to a *.\em c or *.\em h file
00237 <tr>
00238 <td> `-g` *or* `--geany-mode`
00239 <td> ignored
00240 <td> ignored
00241 <td> ignored
00242 <tr>
00243 <td> `-l` *or* `--list-mode` <td> load all *.\em bas and *.
00244     \em bi files, emit output to one file named \em fb-doc.lfn
00245 <td> load this one file, scan the source tree and load each
00246     \#`INCLUDE` file, emit output to one file named \em fb-doc.lfn</td>
00247 <td> load all files matching the pattern (or multiple patterns), emit
00248     output to one file named \em fb-doc.lfn</td>
00249 </table>
00250
00251 \note A file name may be prepended by a relative or an absolute path.
00252
00253 \note File names or paths including a space character have to be
00254       enclosed by single or double quotes.
00255
00256 \note Multiple file names get separated by a space character or by a
00257       ; character.
00258
00259 \note You can list several file names and patterns in the command line.
00260
00261 \note A file name cannot start with a minus character and cannot
00262       include single or double quote characters.
00263
00264 \note On UNIX like systems usually the shell (bash) expands the file
00265       patterns (fb-doc doesn't see the pattern, but gets the files
00266       list instead, so option `--recursive` doesn't work -- enclose
00267       the pattern by single or double quotes to hinder that).
00268
00269
00270 \section sectTabInterForm Intermediate Format
00271
00272 The main trick to make a C parsers work on FB source is to transform
00273 the declarations for variables and functions. FB syntax is full of
00274 keywords for better human readability. Most of them are unknown for the
00275 C parser (fbc-0.90 has more than 400 keyword, in C it's less than 50).
00276
00277 And the C parsers expect just a single word for a type declaration. But
00278 when we pack all FB information in a single word type name, the C tool
00279 can  handle this syntax and build the documentation output.
00280
00281 Therefor the FB keywords get mangled in to one single word, separated
00282 by an underscore character. The following table contains some examples.
00283 This mangling can get suppressed by option `--cstyle`, to get real C
00284 types emitted. There's also some influences on the emitted TYPEs and
00285 file names of include files.
00286
00287 |                                FB source |            default                | `--cstyle`            |
```

```
00288 | ----------------------------------------: | :--------------------------------: | :--------------------- |
00289 |                       'DIM AS INTEGER Nam' | 'INTEGER Nam'                      | 'int Nam'              |
00290 |                       'DIM Nam AS INTEGER' | 'INTEGER Nam'                      | 'int Nam'              |
00291 |                     'CONST Nam AS INTEGER' | 'CONST_AS_INTEGER Nam'             | 'const int Nam'        |
00292 |                    'Extern Nam AS INTEGER' | 'Extern_AS_INTEGER Nam'            | 'extern int Nam'       |
00293 | 'DECLARE PROPERTY Cu.Tok() AS INTEGER PTR' | 'PROPERTY_AS_INTEGER_PTR Cu.Tok()' | 'int *Cu.Tok(void)'    |
00294 |                  'BYVAL Export_ AS EmitFunc' | 'BYVAL_AS_EmitFunc Export_'        | '
      EmitFunc *Export_'   |
00295 |                           'byref Z as byte' | 'byref_as_byte Z'                  | 'char *Z'              |
00296 |                           'byval Z as byte' | 'byval_as_byte Z'                  | 'char Z'               |
00297 |                     'TYPE Udt ... END TYPE' | 'class Udt{ ... };'                | 'typedef Udt{ ... };' |
00298 |                   \#'INCLUDE ONCE "abc.bi"' | \#'include "abc.bi"'               | \#'include "abc.h"'    |
00299 |                  \#'INCLUDE ONCE "xyz.bas"' | \#'include "xyz.bas"'              | \#'include "xyz.c"'    |
00300
00301
00302 \section sectTabFiles Files
00303
00304 Here's an overview on the important files in the archive *fb-doc.zip*.
00305
00306 The folder *src* contains the source code files of fb-doc and some
00307 additional text for this documentation in the file *Tutorial.bi*.
00308
00309 |        fb-doc/src | Function                                                                |
00310 | ----------------: | :---------------------------------------------------------------------- |
00311 |        fb-doc.bas | The main source code to compile (\#'INCLUDE's other files).             |
00312 |       Tutorial.bi | The text of this Tutorial (no FB source in there).                      |
00313 |        Plugin.bas | The source code for an external emitter (example).                      |
00314 |  other *.bas *.bi | The source code.                                                        |
00315
00316 The folder *doc* is to build this documentation by executing Doxygen in
00317 this folder, check its manual for details. If you want caller / callee
00318 graphs in the documentation you have to install the *dot* tool from
00319 the *GraphViz* package and you have to execute 'fb-doc -l' before you
00320 start doxygen. For repairing the syntax highlighting in the Doxygen
00321 output run 'fb-doc -s' afterwards in this folder.
00322
00323 |        fb-doc/doc | Function                                                                |
00324 | ----------------: | :---------------------------------------------------------------------- |
00325 |          Doxyfile | A configuration file, controlling the Doxygen and fb-doc operations.    |
00326 | DoxygenLayout.xml | A configuration file, controlling the index of these *html* pages.      |
00327 |        fb-doc.lfn | A list of function names generated by executing 'fb-doc -l'.            |
```

15.47 src/Usage.md File Reference

15.48 src/Usage.md

```
00001 How To Use  {#pageUsage}
00002 ==========
00003 \tableofcontents
00004
00005 This is a brief introduction in auto-generating documentation extracted
00006 from the source code. Therefor you'll need well written source code in
00007 correct syntax for the compiler (FB syntax), but also comments matching
00008 the syntax of the back-end in use.
00009
00010 So to test fb-doc and to learn about its usage and features you'll need
00011
00012 -# source code to work on (ie the fb-doc code in the 'src' folder)
00013 -# an executable of fb-doc (ie compiled by 'fbc fb-doc.bas')
00014 -# a documentation back-end (fb-doc comments are formated for Doxygen)
00015 -# optional a GUI frontend (ie Doxywizard)
00016 -# optional further tools (ie *GraphViz* (graphs), *LaTeX* (pdf), ...)
00017
00018
00019
00020
00021
00022
00023
00024 both, an installation of a C tool-chain and some source code with
00025 documentation comments in the tool-chain's syntax.
00026
00027 The easiest way to get started is to use Doxygen and the fb-doc
00028 sourc code. In that case you can skip the next section.
00029
00030
00031 \section sectUsageTemplates Generating Templates
00032
00033 For Doxygen back-end there's no need for generating templates. It's
00034 best practise to write the documenting comment right in front of or
00035 just behind the relevant construct. The only exeption is a
```

```
00036 documenting comment for a function. It contains the names of the
00037 variables in the parameter list.
00038
00039 Its different when using gtk-doc tool-chain. Here the documentation
00040 context is collected in blocks. Such a block contains the name of
00041 the relevant construct (ie a ENUM block) and the names of all its
00042 members, followed by its decription. (Doxygen can handle such blocks
00043 as well, but it isn't well supported.)
00044
00045 Generating such a comment block is a reasonable amount of word. Each
00046 name has to get copied from the source in to the block. fb-doc can
00047 do this for us.
00048
00049 When we use Geany IDE, we can use a convenient method by installing
00050 fb-doc as Geany custom command (see \ref sectInstallGeany) and choose
00051 the emitter for the tool-chain in use (\em GtkDocTemplates or \em
00052 DoxygenTemplates). After loading the source, we select a block of
00053 code, send it to fb-doc and we receive the original block, prepended
00054 by a customized documentation block. We just need to edit the
00055 entries (marked by the text `FIXME`). See \ref subsectExaGtkdoc and
00056 \ref subsectExaDoxy for examples.
00057
00058 We can select a single construct and generate the comment blocks
00059 one-by-one. Or we select a bunch of statements and fb-doc inserts
00060 the templates inbetween the constructs. In case of a block
00061 construct (ENUM / UNION / TYPE) it's advantageous to select the
00062 complete block up to the END ... statement, to get all members
00063 listed in the documentation block. In case of a function
00064 decalaration the selection need not include the function body, but
00065 should contain the complete decalaration.
00066
00067 It's a bit less convenient to auto-generate templates when an other
00068 IDE is in use. In that case the documentation blocks can get
00069 generated for a complete file, by piping its context to fb-doc. See
00070 \ref sectUsagePipe for details.
00071
00072
00073 \section sectUsageGtk Gtk-doc
00074
00075 When using gtk-doc as back-end the documentation comment blocks are
00076 placed before a structure, union, function ... and the names of the
00077 members or parameters are listed in the comment block, prepended by
00078 a \@ character and appended by a colon. See the manual
00079 http://developer.gnome.org/gtk-doc-manual/stable/index.html for
00080 details. Find an example for a gtk-doc documentation comment in \ref
00081 subsectExaGtkdoc.
00082
00083 fb-doc helps us by creating such documentation comment blocks (=
00084 template). We select a piece of code in Geany and send it to the
00085 fb-doc filter. fb-doc creates a template, evaluates the names from
00086 our source code and includes them in the template, followed by the
00087 text `FIXME`. We just replace this text by the proper description.
00088 To use this feature we have to install fb-doc as Geany filter, see
00089 \ref sectInstallGeany for details.
00090
00091 When our documentation texts are done, we create a pseudo C source
00092 tree in a separate folder and use the gtk-doc tool chain on this C
00093 files. Therefor we start a command line interpreter (shell) and
00094 switch to the folder where our FB source is placed. Then, we execute
00095
00096 ~~~{.sh}
00097 fb-doc --asterix --file-mode
00098 ~~~
00099
00100 This makes fb-doc creating a new folder <em>../doc/c_src</em> (if
00101 not present) and write pseudo C files similar to our FB source
00102 (overriding existing files). *.\em bas files get translated to *.\em
00103 c and *.\em bi files get translated to *.\em h. Our documentation
00104 comments get transfered to this files and some of our FB code gets
00105 translated to C syntax. Then we start the gtk-doc tool-chain in the
00106 <em>../doc/c_src</em> folder as descibed in the above mentioned
00107 manual (\em gtkdoc-scan, ...).
00108
00109 gtk-doc works well to document a library API. Especially when it's
00110 related to GLib, GTK+ or gnome software. One of the unique features
00111 is auto-extracting the properties from the source code (I'll publish
00112 an FB tool soon). But it's not the best choise when we want to
00113 document a program like fb-doc. In that case we better use Doxygen.
00114
00115
00116 \section sectUsageDoxy Doxygen Back-End
00117
00118 For Doxygen we can either use separate comment blocks, similar to
00119 the gtk-doc blocks. Doxygen blocks use a different syntax and they
00120 don't need to be placed in front of the related construct. fb-doc
00121 also supports Doxygen templates, but this way of documenting isn't
00122 very common and not well supported yet.
```

```
00123
00124 The other case is &mdash; a more commen way &mdash; to place the
00125 documentation directly in front of (or behind the) related construct.
00126 Rather than documenting a complete block at once, just the class
00127 name or just one member gets documented in one comment. This way is
00128 more convenient, we need not extract the names from our source code.
00129 We just place the documentation beside the construct and Doxygen
00130 picks the names from the source. For that way of documenting there's
00131 no need to generate templates. The only execption is a parameter
00132 list, where the parameter names have to be listed in the
00133 documentation block. See
00134
00135  - the manual
00136    http://www.stack.nl/~dimitri/doxygen/manual.html for details.
00137  - \ref subsectExaDoxy for an example for a function comment block
00138  - all fb-doc source code as further examples.
00139
00140 For Doxygen we can also create the pseudo C files in the
00141 <em>../doc/c_src</em> folder, as described in the previous section.
00142 But it's more convenient to send the pseudo C code directly to
00143 Doxygen without creating intermediate files. This can be done by
00144 using fb-doc as a Doxygen filter. (We use Doxygen as if we were
00145 working on C source code, fb-doc translates on-the-fly when Doxygen
00146 reads the input.)
00147
00148 Doxygen gets controlled by a configuration file. This file contains
00149 all options. Here we can specify the fb-doc filter by adapting the
00150 following:
00151
00152 First we have to tell Doxygen that it should load FB source files:
00153
00154 \verbatim
00155 FILE_PATTERNS          = *.bi \
00156                          *.bas
00157 \endverbatim
00158
00159 and second we need to set fb-doc as the filter for these files
00160
00161 \verbatim
00162 FILTER_PATTERNS        = *.bas=fbdoc \
00163                          *.bi=fbdoc
00164 \endverbatim
00165
00166 \note Doxygen doesn't allow to send additional options directly to
00167       a filter. So if we need further options we can set the name
00168       of a batch file here and call fb-doc from the batch file
00169       passing further options.
00170
00171 That's it. When we now start doxygen with proper settings for the
00172 paths and output format, we get our first auto-created
00173 documentation. (Try it with the fb-doc source code.)
00174
00175 To get caller and callees graphs (as in this document) it gets a
00176 little tricky. Doxygen needs the function calls inside the function
00177 bodies (in the C source). But when fb-doc acts as a Doxygen filter
00178 it gets restarted on each file. So it doesn't know a function
00179 declaration in file B when working on file A. The names of all the
00180 functions must be known to evaluate their calls in a function body
00181 and to generate proper pseudo C source. That's why fb-doc reads the
00182 names from a file called <em>fb-doc.lfn</em> in the current folder.
00183
00184 We needn't generate this file manually, fb-doc can do this for
00185 us by executing the command
00186
00187 ~~~{.sh}
00188 fb-doc --list-mode
00189 ~~~
00190
00191 in our source folder. This writes a new (or overrides an existing) file
00192 <em>fb-doc.lfn</em> in the current folder. When we don't start Doxygen
00193 in our source folder, we have to move the file to the folder where
00194 we execute the \em doxygen command. When we use Doxywizzard (the GUI
00195 frontend for Doxygen) this is the folder specified under
00196
00197 \verbatim
00198 Step1: Specify the working directory from which doxygen will run
00199 \endverbatim
00200
00201 And we have to specify fb-doc as filter for source files and enable
00202 source file filtering
00203
00204 \verbatim
00205 FILTER_SOURCE_FILES    = YES
00206
00207 FILTER_SOURCE_PATTERNS = *.bas=fbdoc \
00208                          *.bi=fbdoc
00209 \endverbatim
```

```
00210
00211 In the next run Doxygen should be able to generate caller and callee
00212 graphs in the documentation (if the settings are correct and all
00213 tools are available).
00214
00215 But this has a downside: when we want the source files in the
00216 documentation (as in this document) the C-like code is used. To get
00217 the FB code we have to run Doxygen twice:
00218
00219  -# without source filtering to get the \em html files with FB source
00220
00221  -# with source filtering to get the graphs
00222
00223 Before the second run we save the \em html source files in a
00224 separate folder by executing in the Doxygen html output folder
00225 something like (example for LINUX OS)
00226
00227 ~~~{.sh}
00228 mkdir bas
00229 mv *bas_source.html bas
00230 mv *bi_source.html bas
00231 ~~~
00232
00233 and after the second run we restore the FB source files
00234
00235 ~~~{.sh}
00236 rm *bas_source.html
00237 rm *bi_source.html
00238 cd bas
00239 mv * ..
00240 cd ..
00241 rmdir bas
00242 ~~~
00243
00244
00245 \section sectUsageLibs C Headers
00246
00247 Since fb-doc needs to generate C-like code for the back-end parsers,
00248 it can also be useful to create a set of C headers for a library
00249 written in FreeBasic. By default the FB types get mangled in to one
00250 word (for documentation purposes). Ie a FB parameter <em>byref Nam
00251 as const short</em> gets the pseudo C code <em>byref_as_const_short
00252 Nam</em>. This mangling can get suppressed by option `--cstyle`
00253 and fb-doc emitts real C types. In that case the example gets
00254 <em>const short* Nam</em> and this code can be used by a C compiler.
00255
00256 So when we need C headers for our FB library we just execute
00257
00258 ~~~{.sh}
00259 fb-doc --file-mode --cstyle "*.bi"
00260 ~~~
00261
00262 in our source folder and get a set of C translations in *.\em h
00263 files in the target folder (<em>../doc/c_src</em> by default).
00264
00265 \note fb-doc doesn't handle initializers. You have to check them
00266       manually.
00267
00268
00269 \section sectUsagePipe Piping
00270
00271 By default fb-doc reads its input for (one or more) files and sends
00272 its output to STDOUT. We can switch to STDIN input by option \em
00273 --geany-mode and then use the operating system commands to control the
00274 data flow in and from fb-doc.
00275
00276 Ie this is useful when we don't use Geany IDE or when we want to get
00277 templates for a complete file or a bunch of files at once. We can
00278 pipe the files context to the fb-doc STDIN channel and the fb-doc
00279 output to a file.
00280
00281 To get templates for all constructs in an existing file, first we
00282 create a copy the file (example for LINUX OS)
00283
00284 ~~~{.sh}
00285 cp filename.bas filename.txt
00286 ~~~
00287
00288 Then we pipe the copy to fb-doc in geany mode (using emitter \em
00289 DoxygenTemplates here). The output gets piped to the original file
00290 (omit <em>>filename.bas</em> to see the output in the console)
00291
00292 ~~~{.sh}
00293 fbdoc --geany-mode doxy < filename.txt > filename.bas
00294 ~~~
00295
00296 Finally we may want to delete the intermediate file
```

```
00297
00298 ~~~{.sh}
00299 rm filename.txt
00300 ~~~
00301
00302 To insert templates in all files of an existing project we write a
00303 batch file (or FB program) that does these three steps for all our
00304 source files.
```

Index

~Doxyfile
    Doxyfile, 56
~Options
    Options, 82

A
    Parser, 131
AdLine
    Options, 91
add
    RepData, 132
addPath
    Options, 88
AliTok
    Parser, 129
AllCallees
    Options, 91
As_Tok
    Parser, 129
Asterix
    Options, 91

BitIni
    Parser, 104
BitTok
    Parser, 128
BlockNam
    Parser, 127
Buf
    Parser, 127
Buffer
    Doxyfile, 57
By_Tok
    Parser, 129

C_SOURCE
    Options, 81
C_STYLE
    Options, 81
CASE_LOWER
    Options, 82
CASE_MIXED
    Options, 82
CASE_ORIGN
    Options, 82
CASE_UPPER
    Options, 82
c_Block
    fb-doc_emitters.bas, 223
c_CTOR
    fb-doc_emitters.bas, 226
c_Init
    fb-doc_emitters.bas, 225
c_decl_
    fb-doc_emitters.bas, 221
c_defi_
    fb-doc_emitters.bas, 220
c_error
    fb-doc_emitters.bas, 224
c_exit
    fb-doc_emitters.bas, 225
c_func_
    fb-doc_emitters.bas, 220
c_include
    fb-doc_emitters.bas, 219
CALLEE_TR
    fb-doc_emit_callees.bas, 159
CALLEES_FILE
    fb-doc_options.bi, 251
cArrDim
    fb-doc_emitters.bas, 212
cCreateFunction
    fb-doc_emitters.bas, 217
cCreateTypNam
    fb-doc_emitters.bas, 214
cEmitComments
    fb-doc_emitters.bas, 209
cEmitSource
    fb-doc_emitters.bas, 226
cEntryBlockENUM
    fb-doc_emitters.bas, 222
cEntryBlockTypeUnion
    fb-doc_emitters.bas, 224
cEntryListParameter
    fb-doc_emitters.bas, 218
cIni
    fb-doc_emitters.bas, 212
CMNT_A
    Highlighter, 76
cNam
    fb-doc_emitters.bas, 211
COMM_END
    fb-doc.bi, 152
CTOR_
    EmitterIF, 64
CalTok
    Parser, 129
callees_class_
    fb-doc_emit_callees.bas, 157

callees_decl_
fb-doc_emit_callees.bas, 156
callees_func_
fb-doc_emit_callees.bas, 157
callees_include
fb-doc_emit_callees.bas, 158
CaseMode
Options, 89
CaseModes
Options, 81
checkDir
Options, 87
chooseEmitter
Options, 83
Clas_
EmitterIF, 61
Co1Tok
Parser, 128
Co2Tok
Parser, 129
Code
fb-doc_parser.bi, 273
cppCreateFunction
fb-doc_emitters.bas, 215
cppCreateTypNam
fb-doc_emitters.bas, 213
cppEntryListParameter
fb-doc_emitters.bas, 216
cppNam
fb-doc_emitters.bas, 210
CreateFunction
Options, 92
CreateVariable
Options, 92
CurTok
Parser, 104

DEF_MODE
Options, 81
DOXYGEN_TEMPLATES
Options, 81
DECLARE_
Parser, 124
DEFINE_
Parser, 126
DOXY_END
fb-doc_emit_doxy.bas, 167
DOXY_START
fb-doc_emit_doxy.bas, 167
DTOR_
EmitterIF, 64
Decl_
EmitterIF, 59
Defi_
EmitterIF, 62
demuxDecl
Parser, 118
demuxNam
Parser, 116
demuxTyp
Parser, 117
DimTok
Parser, 128
DirUp
Options, 90
DivTok
Parser, 129
dll_CTOR
Plugin.bas, 292
dll_DTOR
Plugin.bas, 292
dll_class
Plugin.bas, 291
dll_declare
Plugin.bas, 289
dll_define
Plugin.bas, 291
dll_empty
Plugin.bas, 292
dll_enum
Plugin.bas, 290
dll_error
Plugin.bas, 291
dll_exit
Plugin.bas, 292
dll_function
Plugin.bas, 290
dll_include
Plugin.bas, 291
dll_init
Plugin.bas, 291
dll_union
Plugin.bas, 291
DllEmitter
Options, 90
do_files
Highlighter, 70
doDoxy
Highlighter, 69
doFile
Options, 86
Doxy
Doxyfile, 57
doxy_Block
fb-doc_emit_doxy.bas, 166
doxy_decl_
fb-doc_emit_doxy.bas, 163
doxy_defi_
fb-doc_emit_doxy.bas, 164
doxy_emitBlockNames
fb-doc_emit_doxy.bas, 165
doxy_empty
fb-doc_emit_doxy.bas, 167
doxy_entryListPara
fb-doc_emit_doxy.bas, 162
doxy_func_
fb-doc_emit_doxy.bas, 162

DoxyFiles
Highlighter, 75
Doxyfile, 55
~Doxyfile, 56
Buffer, 57
Doxy, 57
Doxyfile, 56
Errr, 57
Length, 57
Tag, 56

ERROR_MESSAGE
Options, 80
EXTERNAL
Options, 81
ENUM_
Parser, 121
ERROUT
fb-doc_options.bi, 251
EXIT_STOP
fb-doc_parser.bas, 256
Efnr
Options, 91
Emit
Parser, 131
EmitFunc
fb-doc_emitters.bi, 240
EmitIF
Options, 90
EmitTyp
Options, 89
EmitterIF, 57
CTOR_, 64
Clas_, 61
DTOR_, 64
Decl_, 59
Defi_, 62
Empty_, 63
Enum_, 60
Error_, 63
Exit_, 64
Func_, 60
Incl_, 62
Init_, 59
Nam, 64
Unio_, 61
EmitterInit
Plugin.bas, 292
EmitterTypes
Options, 81
Emitters
fb-doc_emitters.bi, 240
Empty_
EmitterIF, 63
EndTok
Parser, 130
Enum_
EmitterIF, 60
EoS_Modi
Parser, 99
eol
Highlighter, 74
ErrMsg
Parser, 127
Error_
EmitterIF, 63
Errr
Doxyfile, 57
Options, 90
Parser, 115
Exit_
EmitterIF, 64

FB_CODE
Highlighter, 67
FB_KEYW
Highlighter, 67
FB_KWFL
Highlighter, 67
FB_KWTP
Highlighter, 67
FB_PREP
Highlighter, 67
FB_STYLE
Options, 81
FB_SYMB
Highlighter, 67
FILE_MODE
Options, 81
FUNCTION_NAMES
Options, 81
FBDOC_BINARY
fb-doc.bi, 151
FBDOC_MARK
Highlighter, 76
FBDOC_TARGET
fb-doc.bi, 151
FIXME
fb-doc.bi, 152
FUNCTION_
Parser, 123
fb-doc.bas
main, 147
PROG_NAME, 147
PROG_VERS, 147
fb-doc.bi
COMM_END, 152
FBDOC_BINARY, 151
FBDOC_TARGET, 151
FIXME, 152
MSG_HELP, 150
MSG_VERSION, 151
MSG_WELCOME, 151
fb-doc_emit_callees.bas
CALLEE_TR, 159
callees_class_, 157
callees_decl_, 156
callees_func_, 157

callees_include, 158
writeLFN, 156
fb-doc_emit_doxy.bas
DOXY_END, 167
DOXY_START, 167
doxy_Block, 166
doxy_decl_, 163
doxy_defi_, 164
doxy_emitBlockNames, 165
doxy_empty, 167
doxy_entryListPara, 162
doxy_func_, 162
fb-doc_emit_gtk.bas
GTK_END, 178
GTK_START, 178
gtk_Block, 176
gtk_decl_, 173
gtk_defi_, 172
gtk_emit_Name, 172
gtk_emitBlockNames, 175
gtk_empty, 177
gtk_func_, 174
SINCE, 178
fb-doc_emit_syntax.bas
GET_WORD_TYPE, 183
html_eol, 184
html_specials, 183
synt_exit, 187
synt_func_, 188
synt_incl, 187
synt_init, 186
tex_eol, 185
tex_specials, 184
xml_eol, 186
xml_specials, 184
fb-doc_emitters.bas
c_Block, 223
c_CTOR, 226
c_Init, 225
c_decl_, 221
c_defi_, 220
c_error, 224
c_exit, 225
c_func_, 220
c_include, 219
cArrDim, 212
cCreateFunction, 217
cCreateTypNam, 214
cEmitComments, 209
cEmitSource, 226
cEntryBlockENUM, 222
cEntryBlockTypeUnion, 224
cEntryListParameter, 218
cIni, 212
cNam, 211
cppCreateFunction, 215
cppCreateTypNam, 213
cppEntryListParameter, 216
cppNam, 210
geanyExit, 228
geanyInit, 227
LOFN, 228
fb-doc_emitters.bi
EmitFunc, 240
Emitters, 240
null_emitter, 240
Parser_, 240
WITH_NEW_EMITTER, 240
fb-doc_options.bi
CALLEES_FILE, 251
ERROUT, 251
MSG_END, 251
MSG_LINE, 251
OPT, 251
fb-doc_parser.bas
EXIT_STOP, 256
SCAN_ML_COMM, 255
SCAN_QUOTE, 255
SCAN_SL_COMM, 255
SCAN_WORD, 256
SETOK, 256
SKIP, 255
fb-doc_parser.bi
Code, 273
NL, 273
SLASH, 273
FbFiles
Highlighter, 74
FbPath
Highlighter, 74
File_
Parser, 100
FileIncl
Options, 90
FileModi
Options, 85
FileName
Options, 90
Fin
Parser, 130
FunTok
Parser, 129
Func_
EmitterIF, 60

GEANY_MODE
Options, 81
GTK_DOC_TEMPLATES
Options, 81
GET_WORD_TYPE
fb-doc_emit_syntax.bas, 183
GTK_END
fb-doc_emit_gtk.bas, 178
GTK_START
fb-doc_emit_gtk.bas, 178
geanyExit
fb-doc_emitters.bas, 228

geanyInit
fb-doc_emitters.bas, 227
generate_all
Highlighter, 72
generate_code
Highlighter, 73
getToken
Parser, 114
gtk_Block
fb-doc_emit_gtk.bas, 176
gtk_decl_
fb-doc_emit_gtk.bas, 173
gtk_defi_
fb-doc_emit_gtk.bas, 172
gtk_emit_Name
fb-doc_emit_gtk.bas, 172
gtk_emitBlockNames
fb-doc_emit_gtk.bas, 175
gtk_empty
fb-doc_emit_gtk.bas, 177
gtk_func_
fb-doc_emit_gtk.bas, 174

HELP_MESSAGE
Options, 80
Highlighter, 65
CMNT_A, 76
do_files, 70
doDoxy, 69
DoxyFiles, 75
eol, 74
FB_CODE, 67
FB_KEYW, 67
FB_KWFL, 67
FB_KWTP, 67
FB_PREP, 67
FB_SYMB, 67
FBDOC_MARK, 76
FbFiles, 74
FbPath, 74
generate_all, 72
generate_code, 73
Highlighter, 68
HtmlPath, 75
HtmlSuff, 75
Ifnr, 77
InPath, 75
KEYW_A, 76
KWFL_A, 76
KWTP_A, 76
LastLine, 75
LineNo, 77
PREP_A, 76
Pars, 75
prepare, 74
prepare_html, 72
prepare_tex, 71
prepare_xml, 71
QUOT_A, 76
QUOT_E, 76
SPAN_E, 76
special_chars, 74
Symbols, 75
TexPath, 75
word_type, 74
WordTypes, 67
XmlPath, 75
Highlighter.__unnamed__, 182
Highlighter.__unnamed__.__unnamed__, 182
html_eol
fb-doc_emit_syntax.bas, 184
html_specials
fb-doc_emit_syntax.bas, 183
HtmlPath
Highlighter, 75
HtmlSuff
Highlighter, 75

I
RepData, 134
INCLUDE_
Parser, 125
Ifnr
Highlighter, 77
InFiles
Options, 90
InPath
Highlighter, 75
Parser, 127
InRecursiv
Options, 91
InTree
Options, 91
Parser, 130
Incl_
EmitterIF, 62
Include
Parser, 102
IniTok
Parser, 128
Init_
EmitterIF, 59

JoComm
Options, 91

KEYW_A
Highlighter, 76
KWFL_A
Highlighter, 76
KWTP_A
Highlighter, 76

L
Parser, 131
LIST_MODE
Options, 81
LOFN

fb-doc_emitters.bas, 228
LastLine
Highlighter, 75
Length
Doxyfile, 57
Level
Options, 91
LevelCount
Parser, 130
LineNo
Highlighter, 77
Parser, 130
ListCount
Parser, 131

MSG_ERROR
Parser, 97
MSG_STOP
Parser, 97
MACRO_
Parser, 126
MSG_END
fb-doc_options.bi, 251
MSG_HELP
fb-doc.bi, 150
MSG_LINE
fb-doc_options.bi, 251
MSG_VERSION
fb-doc.bi, 151
MSG_WELCOME
fb-doc.bi, 151
main
fb-doc.bas, 147

NL
fb-doc_parser.bi, 273
Nam
EmitterIF, 64
NamTok
Parser, 128
null_emitter
fb-doc_emitters.bi, 240

O
RepData, 134
OPT
fb-doc_options.bi, 251
Ocha
Options, 91
Options, 77
∼Options, 82
AdLine, 91
addPath, 88
AllCallees, 91
Asterix, 91
C_SOURCE, 81
C_STYLE, 81
CASE_LOWER, 82
CASE_MIXED, 82
CASE_ORIGN, 82
CASE_UPPER, 82
CaseMode, 89
CaseModes, 81
checkDir, 87
chooseEmitter, 83
CreateFunction, 92
CreateVariable, 92
DEF_MODE, 81
DOXYGEN_TEMPLATES, 81
DirUp, 90
DllEmitter, 90
doFile, 86
ERROR_MESSAGE, 80
EXTERNAL, 81
Efnr, 91
EmitIF, 90
EmitTyp, 89
EmitterTypes, 81
Errr, 90
FB_STYLE, 81
FILE_MODE, 81
FUNCTION_NAMES, 81
FileIncl, 90
FileModi, 85
FileName, 90
GEANY_MODE, 81
GTK_DOC_TEMPLATES, 81
HELP_MESSAGE, 80
InFiles, 90
InRecursiv, 91
InTree, 91
JoComm, 91
LIST_MODE, 81
Level, 91
Ocha, 91
Options, 82
OutPath, 90
Pars, 89
parseCLI, 82
parseOptpara, 83
RunMode, 89
RunModes, 80
SYNT_MODE, 81
SYNTAX_REPAIR, 81
scanFiles, 88
StartPath, 90
Types, 89
TypesStyle, 81
VERSION_MESSAGE, 81
OutPath
Options, 90

PREP_A
Highlighter, 76
PROG_NAME
fb-doc.bas, 147
PROG_VERS
fb-doc.bas, 147

ParTok
Parser, 129
Pars
Highlighter, 75
Options, 89
parseBlockEnum
Parser, 110
parseBlockTyUn
Parser, 111
parseCLI
Options, 82
parseListNam
Parser, 108
parseListNamTyp
Parser, 108
parseListPara
Parser, 109
parseOptpara
Options, 83
Parser, 92
A, 131
AliTok, 129
As_Tok, 129
BitIni, 104
BitTok, 128
BlockNam, 127
Buf, 127
By_Tok, 129
CalTok, 129
Co1Tok, 128
Co2Tok, 129
CurTok, 104
DECLARE_, 124
DEFINE_, 126
demuxDecl, 118
demuxNam, 116
demuxTyp, 117
DimTok, 128
DivTok, 129
ENUM_, 121
Emit, 131
EndTok, 130
EoS_Modi, 99
ErrMsg, 127
Errr, 115
FUNCTION_, 123
File_, 100
Fin, 130
FunTok, 129
getToken, 114
INCLUDE_, 125
InPath, 127
InTree, 130
Include, 102
IniTok, 128
L, 131
LevelCount, 130
LineNo, 130
ListCount, 131
MSG_ERROR, 97
MSG_STOP, 97
MACRO_, 126
NamTok, 128
ParTok, 129
parseBlockEnum, 110
parseBlockTyUn, 111
parseListNam, 108
parseListNamTyp, 108
parseListPara, 109
Parser, 99
ParserTokens, 97
Po, 130
pre_parse, 112
PtrCount, 130
PtrTok, 128
ShaTok, 128
skipOverBrclo, 119
skipOverColon, 118
skipOverComma, 119
SrcBgn, 130
StaTok, 128
StdIn, 101
SubStr, 105, 106
TO_COLON, 99
TO_END, 99
TO_END_BLOCK, 99
TO_EOL, 99
TOK_ABST, 98
TOK_ALIA, 98
TOK_AS, 98
TOK_BRCLO, 97
TOK_BROPN, 97
TOK_BYRE, 98
TOK_BYTE, 99
TOK_BYVA, 98
TOK_CDEC, 98
TOK_CLAS, 98
TOK_COMM, 98
TOK_COMMA, 97
TOK_COMML, 99
TOK_COMSL, 99
TOK_CONS, 98
TOK_CTOR, 98
TOK_DECL, 98
TOK_DEFI, 98
TOK_DIM, 98
TOK_DOT, 97
TOK_DOUB, 99
TOK_DTOR, 98
TOK_EMAC, 98
TOK_END, 98
TOK_ENUM, 98
TOK_EOS, 97
TOK_EQUAL, 97
TOK_EXDS, 98
TOK_EXPO, 98

TOK_EXRN, 98
TOK_FILD, 98
TOK_FUNC, 98
TOK_INCL, 98
TOK_INT, 99
TOK_KLCLO, 97
TOK_KLOPN, 97
TOK_LATTE, 97
TOK_LIB, 98
TOK_LINT, 99
TOK_LONG, 99
TOK_MACR, 98
TOK_NAMS, 98
TOK_ONCE, 98
TOK_OPER, 98
TOK_OVER, 98
TOK_PASC, 98
TOK_PRIV, 98
TOK_PROP, 98
TOK_PROT, 98
TOK_PTR, 98
TOK_PUBL, 98
TOK_QUOTE, 97
TOK_RDIM, 98
TOK_SCOP, 98
TOK_SHAR, 98
TOK_SHOR, 99
TOK_SING, 99
TOK_STAT, 98
TOK_STCL, 98
TOK_STRI, 99
TOK_SUB, 98
TOK_TRIDO, 97
TOK_TYPE, 98
TOK_UBYT, 99
TOK_UINT, 99
TOK_ULIN, 99
TOK_ULNG, 99
TOK_UNIO, 98
TOK_USHO, 99
TOK_VAR, 98
TOK_VIRT, 98
TOK_WORD, 99
TOK_WSTR, 99
TOK_ZSTR, 99
TYPE_, 121
Tk, 131
Tk1, 129
ToLast, 131
Tok, 127
tokenize, 114
TypTok, 128
UNION_, 122
USubStr, 113
UserTok, 130
VAR_, 120
writeOut, 103
Parser_
fb-doc_emitters.bi, 240
ParserTokens
Parser, 97
Plugin.bas
dll_CTOR, 292
dll_DTOR, 292
dll_class, 291
dll_declare, 289
dll_define, 291
dll_empty, 292
dll_enum, 290
dll_error, 291
dll_exit, 292
dll_function, 290
dll_include, 291
dll_init, 291
dll_union, 291
EmitterInit, 292
Po
Parser, 130
pre_parse
Parser, 112
prepare
Highlighter, 74
prepare_html
Highlighter, 72
prepare_tex
Highlighter, 71
prepare_xml
Highlighter, 71
PtrCount
Parser, 130
PtrTok
Parser, 128

QUOT_A
Highlighter, 76
QUOT_E
Highlighter, 76

rep
RepData, 133
RepData, 131
add, 132
I, 134
O, 134
rep, 133
RunMode
Options, 89
RunModes
Options, 80

SYNT_MODE
Options, 81
SYNTAX_REPAIR
Options, 81
SCAN_ML_COMM
fb-doc_parser.bas, 255
SCAN_QUOTE

fb-doc_parser.bas, 255
SCAN_SL_COMM
fb-doc_parser.bas, 255
SCAN_WORD
fb-doc_parser.bas, 256
SETOK
fb-doc_parser.bas, 256
SINCE
fb-doc_emit_gtk.bas, 178
SKIP
fb-doc_parser.bas, 255
SLASH
fb-doc_parser.bi, 273
SPAN_E
Highlighter, 76
scanFiles
Options, 88
ShaTok
Parser, 128
skipOverBrclo
Parser, 119
skipOverColon
Parser, 118
skipOverComma
Parser, 119
special_chars
Highlighter, 74
src/Development.md, 135
src/Emitters.md, 135
src/Examples.md, 137
src/Extend.md, 144
src/Files.md, 277
src/Installation.md, 278
src/Introduction.md, 280
src/Options.md, 282
src/Plugin.bas, 288, 292
src/Tables.md, 294
src/Usage.md, 298
src/fb-doc.bas, 146, 148
src/fb-doc.bi, 149, 152
src/fb-doc_doxyfile.bas, 153
src/fb-doc_emit_callees.bas, 155, 159
src/fb-doc_emit_doxy.bas, 160, 168
src/fb-doc_emit_gtk.bas, 171, 178
src/fb-doc_emit_syntax.bas, 181, 189
src/fb-doc_emitters.bas, 207, 228
src/fb-doc_emitters.bi, 238, 241
src/fb-doc_options.bas, 242, 243
src/fb-doc_options.bi, 249, 251
src/fb-doc_parser.bas, 253, 256
src/fb-doc_parser.bi, 272, 273
SrcBgn
Parser, 130
StaTok
Parser, 128
StartPath
Options, 90
StdIn
Parser, 101
SubStr
Parser, 105, 106
Symbols
Highlighter, 75
synt_exit
fb-doc_emit_syntax.bas, 187
synt_func_
fb-doc_emit_syntax.bas, 188
synt_incl
fb-doc_emit_syntax.bas, 187
synt_init
fb-doc_emit_syntax.bas, 186

TO_COLON
Parser, 99
TO_END
Parser, 99
TO_END_BLOCK
Parser, 99
TO_EOL
Parser, 99
TOK_ABST
Parser, 98
TOK_ALIA
Parser, 98
TOK_AS
Parser, 98
TOK_BRCLO
Parser, 97
TOK_BROPN
Parser, 97
TOK_BYRE
Parser, 98
TOK_BYTE
Parser, 99
TOK_BYVA
Parser, 98
TOK_CDEC
Parser, 98
TOK_CLAS
Parser, 98
TOK_COMM
Parser, 98
TOK_COMMA
Parser, 97
TOK_COMML
Parser, 99
TOK_COMSL
Parser, 99
TOK_CONS
Parser, 98
TOK_CTOR
Parser, 98
TOK_DECL
Parser, 98
TOK_DEFI
Parser, 98
TOK_DIM

Parser, 98
TOK_DOT
Parser, 97
TOK_DOUB
Parser, 99
TOK_DTOR
Parser, 98
TOK_EMAC
Parser, 98
TOK_END
Parser, 98
TOK_ENUM
Parser, 98
TOK_EOS
Parser, 97
TOK_EQUAL
Parser, 97
TOK_EXDS
Parser, 98
TOK_EXPO
Parser, 98
TOK_EXRN
Parser, 98
TOK_FILD
Parser, 98
TOK_FUNC
Parser, 98
TOK_INCL
Parser, 98
TOK_INT
Parser, 99
TOK_KLCLO
Parser, 97
TOK_KLOPN
Parser, 97
TOK_LATTE
Parser, 97
TOK_LIB
Parser, 98
TOK_LINT
Parser, 99
TOK_LONG
Parser, 99
TOK_MACR
Parser, 98
TOK_NAMS
Parser, 98
TOK_ONCE
Parser, 98
TOK_OPER
Parser, 98
TOK_OVER
Parser, 98
TOK_PASC
Parser, 98
TOK_PRIV
Parser, 98
TOK_PROP
Parser, 98
TOK_PROT
Parser, 98
TOK_PTR
Parser, 98
TOK_PUBL
Parser, 98
TOK_QUOTE
Parser, 97
TOK_RDIM
Parser, 98
TOK_SCOP
Parser, 98
TOK_SHAR
Parser, 98
TOK_SHOR
Parser, 99
TOK_SING
Parser, 99
TOK_STAT
Parser, 98
TOK_STCL
Parser, 98
TOK_STRI
Parser, 99
TOK_SUB
Parser, 98
TOK_TRIDO
Parser, 97
TOK_TYPE
Parser, 98
TOK_UBYT
Parser, 99
TOK_UINT
Parser, 99
TOK_ULIN
Parser, 99
TOK_ULNG
Parser, 99
TOK_UNIO
Parser, 98
TOK_USHO
Parser, 99
TOK_VAR
Parser, 98
TOK_VIRT
Parser, 98
TOK_WORD
Parser, 99
TOK_WSTR
Parser, 99
TOK_ZSTR
Parser, 99
TYPE_
Parser, 121
Tag
Doxyfile, 56
tex_eol

fb-doc_emit_syntax.bas, 185
tex_specials
fb-doc_emit_syntax.bas, 184
TexPath
Highlighter, 75
Tk
Parser, 131
Tk1
Parser, 129
ToLast
Parser, 131
Tok
Parser, 127
tokenize
Parser, 114
TypTok
Parser, 128
Types
Options, 89
TypesStyle
Options, 81

UNION_
Parser, 122
USubStr
Parser, 113
Unio_
EmitterIF, 61
UserTok
Parser, 130

VERSION_MESSAGE
Options, 81
VAR_
Parser, 120

WITH_NEW_EMITTER
fb-doc_emitters.bi, 240
word_type
Highlighter, 74
WordTypes
Highlighter, 67
writeLFN
fb-doc_emit_callees.bas, 156
writeOut
Parser, 103

xml_eol
fb-doc_emit_syntax.bas, 186
xml_specials
fb-doc_emit_syntax.bas, 184
XmlPath
Highlighter, 75