EurekaLog 7.x.x code reference

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Table of Contents

Foreword 0

Part I Main routines 2

1 IsEurekaLogInstalled ... 2
2 IsEurekaLogActive ... 3
3 IsEurekaLogActiveInThread ... 5
4 SetEurekaLogState ... 6
5 SetEurekaLogStateInThread ... 7
6 CurrentEurekaLogOptions ... 8
7 LastBugID ... 9

Part II Events 11

1 OnRaise ... 12
2 OnExceptionNotify ... 14
3 OnExceptionAction ... 16
4 OnExceptionError ... 18
5 OnPasswordRequest ... 19
6 OnAttachedFilesRequest ... 21
7 OnZippedFilesRequest ... 22
8 OnReproduceRequest ... 23
9 OnCustomFileName ... 24
10 OnCustomBugID ... 25
11 OnCustomDataRequest ... 26
12 OnCustomWebFieldsRequest ... 27
13 OnCustomButtonClick ... 28
14 OnHelpButtonClick ... 29
15 OnBeginReportGen/OnEndReportGen ... 30

Part III Component 33

1 Events ... 34

Part IV Dialogs 36

1 Base class ... 36
2 Null dialog ... 37
3 RTL dialog ... 37
4 Console ... 37
5 MessageBox dialog ... 37
MessageBox detailed dialog ... 37
6 MS Classic ... 37
7 EurekaLog ... 37
EurekaLog detailed ... 37

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8 CGI ... 37
9 ISAPI ... 37
10 IntraWeb ... 37
11 WER ... 37
12 System Log ... 37

Part V Debug information providers 39

1 EurekaLog ... 39
2 .map files ... 39
3 Turbo Debugger (TDS) ... 39
4 JEDI (Jcl) ... 39
5 madExcept ... 39
6 Microsoft ... 39
7 DLL export table ... 39

Part VI Send engines 41

1 mailto ... 41
2 Simple MAPI ... 41
3 MAPI ... 42
4 SMTP ... 42
SMTP server ... 42
SMTP client ... 43
5 HTTP(S) ... 43
6 FTP ... 43
7 Web ... 44
Mantis ... 44
BugZilla ... 44
FogBugz ... 44
8 API ... 44
Mantis ... 44
FogBugz ... 45

Part VII All Units 47

1 EAppCLX ... 53
2 EAppCrossModule ... 53
3 EAppFMX ... 54
4 EAppHTTPApp ... 54
5 EAppType ... 54
Routines ... 55
6 EAppVCL ... 56
7 EAppWebBroker ... 56
8 EBase ... 57
Classes ... 57
Routines ... 58
9 EBaseClasses ... 58
Classes ... 59
Types ... 59

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10 ECallStack ... 59
Classes ... 60
Globals ... 60
Routines ... 60
Types ... 61
11 EClasses ... 61
Classes ... 61
12 EClassUtils ... 61
13 ECompatibility ... 62
14 EComponent ... 62
Classes ... 62
15 EConfig ... 63
Routines ... 63
16 EConsts ... 65
Globals ... 65
17 ECore ... 65
18 EDbgHelp ... 65
19 EDebugDWARF ... 66
20 EDebugEL ... 66
21 EDebugEL7 ... 67
22 EDebugExportBase ... 67
23 EDebugExports ... 67
24 EDebugInfo ... 68
Routines ... 68
Types ... 68
25 EDebugJCL ... 69
26 EDebugMAB ... 69
27 EDebugMAB2 ... 69
28 EDebugMap ... 70
29 EDebugME ... 70
30 EDebugMS ... 70
31 EDebugStabs ... 71
32 EDebugTD32 ... 71
33 EDialog ... 72
Classes ... 73
Routines ... 73
Types ... 73
34 EDialogCGI ... 73
35 EDialogConsole ... 73
36 EDialogHTMLBased ... 74
37 EDialogSend ... 74
38 EDialogSendWinAPI ... 74
39 EDialogService ... 75
40 EDialogWeb ... 75
41 EDialogWER ... 75
42 EDialogWinAPI ... 76

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

43 EDialogWinAPIEurekaLog 76
44 EDialogWinAPIEurekaLogDetailed 76
45 EDialogWinAPIMSClassic 77
46 EDialogWinAPIStepsToReproduce 77
47 EDisAsm 78
Routines 78
Types 78
48 EDLLs 78
49 EDNS 79
Classes 79
Routines 79
50 EDWARF 79
51 EEncoding 79
52 EEncrypt 80
Routines 81
Types 83
53 EEvents 83
Routines 85
Types 87
54 EException 89
Classes 90
55 EExceptionHook 90
56 EExceptionInfo 90
57 EExceptionInfoBCB 91
58 EExceptionInfoCLR 91
59 EExceptionInfoCpp 91
60 EExceptionInfoDelphi2 92
61 EExceptionInfoDelphiANSI 92
62 EExceptionInfoDelphiUnicode 92
63 EExceptionInfoGeneric 92
64 EExceptionInfoLLVM 93
65 EExceptionInfoMSVC 93
66 EExceptionManager 93
Classes 94
Routines 94
Types 94
67 EFileMemory 95
68 EFixSafeCallException 95
69 EFonts 95
Routines 95
70 EFreeze 96
Classes 96
Globals 96
Routines 97
Types 97
71 EHash 97
Routines 98

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Types ... 99
72 EHook ... 99
73 EHTMLHelp ... 99
74 EInfoFormat ... 99
75 EInject ... 100
76 EInterfaces ... 100
77 EJSON ... 100
78 EListView ... 100
79 ELogBuilder ... 101
Classes ... 101
Globals ... 102
Routines ... 102
Types ... 102
80 ELogging ... 103
Interfaces ... 103
Routines ... 103
Types ... 105
81 ELogManager ... 105
82 ELowLevel ... 105
Routines ... 106
83 ELowLevelClasses ... 108
84 ELowLevelExceptions ... 108
85 EMantisConnectStub ... 108
86 EMAPI ... 108
87 EMapParserBase ... 109
Classes ... 109
88 EMapWin32 ... 109
89 EMemLeaks ... 109
Routines ... 111
Types ... 113
90 EMLang ... 113
91 EModules ... 114
Routines ... 114
92 EMonitors ... 114
93 ENT ... 114
94 EOSApiList ... 115
95 EPChars ... 115
96 EPEImage ... 115
97 EPNG ... 115
98 EResLeaks ... 116
Routines ... 116
Types ... 116
99 EResourceStrings ... 116
100 ERunTime ... 116
101 EScreenShot ... 119
Routines ... 119
Types ... 120

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

102 ESend ... 120
Classes ... 121
Routines ... 121
Types ... 121
103 ESendAPIBugZilla ... 121
Classes ... 122
104 ESendAPIExceptionless ... 122
Classes ... 122
105 ESendAPIFogBugz ... 122
Classes ... 123
106 ESendAPIGitHub ... 123
Classes ... 123
107 ESendAPIGitLab ... 124
Classes ... 124
108 ESendAPIJIRA ... 124
Classes ... 124
109 ESendAPIJSON ... 125
110 ESendAPIMantis ... 125
Classes ... 125
111 ESendAPIRedmine ... 126
Classes ... 126
112 ESendAPIYouTrack ... 126
113 ESendMail ... 127
114 ESendMailMAPI ... 127
Classes ... 127
115 ESendMailShell ... 128
Classes ... 128
116 ESendMailSMAPI ... 128
Classes ... 129
117 ESendMailSMTP ... 129
Classes ... 129
118 ESendWeb ... 129
119 ESendWebFTP ... 130
Classes ... 130
120 ESendWebHTTP ... 130
Classes ... 131
121 ESignature ... 131
Routines ... 131
Types ... 131
122 ESOAP ... 132
123 ESOAPMantis ... 132
124 ESocket ... 132
125 ESockets ... 132
126 ESpecificDelphi ... 133
127 ESSLSocket ... 133
128 EStackTracing ... 133
129 EStrConsts ... 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

130 ESysInfo ... 134
Routines ... 134
131 ETaskDialogs ... 136
132 EThreadsManager ... 136
Classes ... 137
Routines ... 137
Types ... 137
133 ETools ... 137
134 ETypes ... 138
Types ... 138
135 EUnmangling ... 139
136 EWCT ... 139
Routines ... 140
137 EWCTSupport ... 140
138 EWebTools ... 140
139 EWER ... 140
Routines ... 141
140 EWinCrypt ... 141
141 EWinSock ... 141
142 ExceptionLog7 ... 141
Routines ... 142
143 EXML ... 143
144 EXMLBuilder ... 143
145 EXMLRPC ... 143
146 EXMLRPCBugZilla ... 144
147 EZip ... 144
Classes ... 144
Globals ... 144
Routines ... 145
Types ... 145
148 EZLib ... 145
149 EZLibHeaders ... 145

Part VIII All Classes 148

1 EDNSError ... 151
2 EZIPException ... 151
3 TBaseDialog ... 151
4 TBaseLogBuilder ... 152
Methods ... 152
Create ... 153
BuildReport ... 153
Refresh ... 154
ToString ... 155
Properties ... 155
FBuilded ... 156
CallStack ... 156
CustomFields ... 157
ExceptionInfo ... 157
Options ... 158

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Report ... 158
5 TConsoleWriterDialog ... 159
6 TDummyLogBuilder ... 159
7 TELAbstractMapParser ... 159
Methods ... 159
DecodeIdent ... 160
MapStringToFileName ... 160
MapStringToModuleName ... 161
MapStringToStr ... 162
PrepareStr ... 163
8 TELDebugInfoExports ... 164
9 TELDebugInfoJedi ... 164
10 TELDebugInfoMadExcept ... 164
11 TELDebugInfoMap ... 164
12 TELDebugInfoNative ... 164
13 TELDebugInfoSource ... 164
14 TELDebugInfoSymbols ... 164
15 TELDebugInfoTD32 ... 165
16 TELFTPSender ... 165
17 TELHTTPSender ... 165
18 TELMailMAPISender ... 165
19 TELMailShellSender ... 166
20 TELMailSimpleMAPISender ... 166
21 TELMailSMTPBaseSender ... 166
22 TELMailSMTPClientSender ... 167
23 TELMailSMTPSender ... 167
24 TELMailSMTPServerSender ... 167
25 TELTrackerBugZillaSender ... 168
26 TELTrackerBugZillaSender ... 168
27 TELTrackerExceptionlessSender ... 168
28 TELTrackerFogBugzSender ... 169
29 TELTrackerFogBugzSender ... 169
30 TELTrackerGitHubSender ... 169
31 TELTrackerGitLabSender ... 170
32 TELTrackerJIRASender ... 170
33 TELTrackerMantisSender ... 170
34 TELTrackerMantisSender ... 171
35 TELTrackerRedmineSender ... 171
36 TELUniversalSender ... 171
37 TEurekaBaseStackList ... 172
Methods ... 173
GetCurrentInstruction ... 173
GetCurrentStackFrame ... 174
GetCurrentStackPointer ... 175
GetCurrentTopOfStack ... 175
Create ... 176

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Build ... 178
38 TEurekaExceptionInfo ... 180
Methods ... 180
ToString ... 180
Properties ... 181
ActiveControl ... 183
ActiveControlClass ... 184
ActiveControlText ... 184
ActiveForm ... 184
ActiveFormClass ... 185
ActiveFormText ... 185
AdditionalInfo ... 186
Address ... 186
BaseException ... 186
BugID ... 187
BugIDSource ... 188
BugIDStr ... 188
CallStack ... 188
Context ... 189
Cursor ... 189
CursorPos ... 189
CustomID ... 190
ExceptionClass ... 190
ExceptionCode ... 190
ExceptionExternal ... 191
ExceptionLocation ... 191
ExceptionMessage ... 192
ExceptionNative ... 192
ExceptionNumber ... 193
ExceptionObject ... 193
ExceptionRecord ... 195
ExceptionThreadID ... 195
Expected ... 195
ExpectedContext ... 196
ExpectedURL ... 197
Handled ... 197
Handler ... 198
Info ... 198
InitializationException ... 199
InnerException ... 199
LastSystemErrorCode ... 200
Module ... 200
ModuleName ... 200
ModuleVersion ... 201
Options ... 201
Processed ... 202
Reader ... 202
ReproduceText ... 203
SafeCallExpt ... 203
Secondary ... 204
StackDump ... 204
StackPoint ... 205
StackTop ... 205
SynchronizeException ... 206
ThreadException ... 206
ThreadID ... 207
Time ... 208
UnhandledExceptionNumber ... 208

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

39 TEurekaLogDetailedDialog ... 208
40 TEurekaLogDialog ... 208
41 TEurekaLogEvents ... 208
Events ... 209
OnAttachedFilesRequest ... 210
OnBeginReportGen ... 211
OnCustomBugID ... 211
OnCustomButtonClick ... 212
OnCustomDataRequest ... 212
OnCustomFileName ... 212
OnCustomWebFieldsRequest ... 213
OnEndReportGen ... 213
OnExceptionAction ... 214
OnExceptionError ... 214
OnExceptionNotify ... 214
OnPasswordRequest ... 215
OnRaise ... 215
OnReproduceRequest ... 215
OnZippedFilesRequest ... 216
42 TEurekaModuleOptions ... 216
43 TEurekaStackListV6 ... 217
44 TEurekaStackListV7 ... 218
45 TEurekaThreadData ... 219
Methods ... 219
Priority ... 220
Properties ... 220
CallerAddress ... 221
CallerFunction ... 222
Handle ... 222
ID ... 223
IsMainThread ... 223
IsRTLThread ... 223
IsWindowsThread ... 224
Kind ... 224
Name ... 225
Parameter ... 225
ParentID ... 226
RealParameter ... 226
RealStartingAddress ... 226
RealStartingFunction ... 227
StackTop ... 227
StartingAddress ... 228
StartingFunction ... 228
Thread ... 229
ThreadClassName ... 229
46 TEurekaThreadDataList ... 229
Properties ... 230
Count ... 230
Items ... 231
ThreadData ... 231
47 TEventLogDialog ... 232
48 TExceptionManager ... 232
Methods ... 232
Instance ... 233
ClearThreadExceptions ... 234

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Handle ... 234
Info ... 237
Notify ... 239
Show LastExceptionData ... 240
StandardEurekaError ... 241
StandardEurekaNotify ... 242
Properties ... 244
ExceptionCount ... 244
LastThreadException ... 244
Mode ... 245
UnhandledExceptionCount ... 245
49 TFreezeThread ... 246
Methods ... 247
ChecksPaused ... 247
Execute ... 248
Properties ... 248
FreezeTimeout ... 248
TargetThreadHandle ... 249
TargetThreadID ... 249
TargetWindow ... 249
50 TGenericStackList ... 249
Methods ... 250
Create ... 250
Properties ... 251
Tracer ... 252
51 TJSONLogBuilder ... 252
52 TLogBuilder ... 253
53 TMainThreadFreezeDetectionThread ... 254
54 TMessageBoxDetailedDialog ... 254
55 TMessageBoxDialog ... 254
56 TMSClassicDialog ... 254
57 TNullDialog ... 254
58 TRTLHandlerDialog ... 254
59 TThreadEx ... 255
Methods ... 258
Create ... 258
Destroy ... 260
Execute ... 261
Properties ... 263
AutoHandleException ... 263
60 TWCTFreezeDetectionThread ... 265
61 TWebDialog ... 265
62 TWebDialog ... 265
63 TWebDialog ... 265
64 TWERDialog ... 265
65 TXMLLogBuilder ... 265
66 TZipFile ... 266
Methods ... 266
ExtractZipFile ... 267
IsValid ... 267
ZipDirectoryContents ... 268
Create ... 268

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Destroy ... 269
Add ... 269
Close ... 271
Extract ... 271
ExtractAll ... 272
IndexOf ... 272
Open ... 273
Read ... 273
Properties ... 274
Comment ... 274
FileComment ... 275
FileCount ... 275
FileInfo ... 275
FileInfos ... 275
FileName ... 276
FileNames ... 276
Mode ... 276

Part IX All Interfaces 278

1 IEurekaLogger ... 278
Methods ... 278
AddCheckPoint ... 280
AddResetSeparator ... 280
AddSeparator ... 280
Category ... 281
Color ... 281
Details ... 282
Enter ... 283
FontColor ... 283
Kind ... 284
Log ... 285
LogAssigned ... 287
LogAssignedPtr ... 287
LogCurrency ... 288
LogDateTime ... 288
LogError ... 288
LogException ... 289
LogIf ... 289
LogIfDateTime ... 290
LogInterval ... 291
LogMemoryAsHex ... 291
LogNote ... 291
LogPointer ... 292
LogReminder ... 292
LogWarning ... 293
LogWindow Handle ... 293
LogWinError ... 294
ResetCheckPoint ... 294
2 IEurekaLoggerParams ... 295
Methods ... 295
C ... 295
D ... 297
P ... 298
Ret ... 300

Part X All Functions 303

1 AcquireThreadData ... 318

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

2 AddCustomDataToBugReport ... 319
3 AddCustomWebField ... 320
4 AllocatePageAlign ... 321
5 AllocationGranularity ... 322
6 AllocMemCountEx ... 322
7 AllocMemCountMax ... 323
8 AllocMemSizeEx ... 324
9 AllocMemSizeMax ... 324
10 BeginThreadEx ... 325
11 BuildBugReport ... 328
12 CanCallMemWipeStack ... 328
13 CheckELMagicCode ... 329
14 CheckHeap ... 330
15 ConfigDelete ... 331
16 ConfigFileName ... 331
17 ConfigIsRegistry ... 333
18 ConfigKeyPath ... 334
19 ConfigReadBool ... 335
20 ConfigReadFloat ... 336
21 ConfigReadInteger ... 337
22 ConfigReadString ... 338
23 ConfigWriteBool ... 339
24 ConfigWriteFloat ... 340
25 ConfigWriteInteger ... 341
26 ConfigWriteString ... 342
27 ConvertAddress ... 343
28 ConvertRetAddr ... 344
29 CRC16Hash ... 345
30 CRC32Hash ... 346
31 CurrentEurekaLogOptions ... 347
32 CurrentInstance ... 348
33 DataProtect ... 349
34 DataUnprotect ... 350
35 DebugBreak ... 352
36 DecodeConfigValue ... 352
37 DecodeInstruction ... 353
38 DisableMemLeaksRegistering ... 354
39 DisasmFunction ... 355
40 DisasmInstruction ... 356
41 DoneCheckFreeze ... 357
42 DumpAllocationsToFile ... 358
43 ELog ... 360

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

44 ELogAddCheckPoint ... 364
45 ELogAddResetSeparator ... 364
46 ELogAddSeparator ... 365
47 ELogAssigned ... 365
48 ELogAssignedPtr ... 366
49 ELogClear ... 367
50 ELogCurrency ... 367
51 ELogDateTime ... 368
52 ELogEnter ... 369
53 ELogError ... 371
54 ELogException ... 371
55 ELogFlush ... 372
56 ELogIf ... 373
57 ELogIfDateTime ... 374
58 ELogInterval ... 375
59 ELogMemoryAsHex ... 376
60 ELogNote ... 377
61 ELogOpen ... 378
62 ELogPointer ... 379
63 ELogReminder ... 380
64 ELogResetCheckPoint ... 380
65 ELogWarning ... 381
66 ELogWindowHandle ... 382
67 ELogWinError ... 383
68 EnableMemLeaksRegistering ... 383
69 EncodeConfigValue ... 384
70 EncryptBufferFree ... 385
71 EndThreadEx ... 386
72 EnsureThreadStack ... 387
73 EnumSenders ... 387
74 EurekaAllocMem ... 388
75 EurekaFreeMem ... 389
76 EurekaGetMem ... 390
77 EurekaReallocMem ... 391
78 EurekaRegisterExpectedMemoryLeak ... 392
79 EurekaRegisterExpectedMemoryLeakDynArray ... 393
80 EurekaRegisterExpectedMemoryLeakStr ... 393
81 EurekaTryGetMem ... 394
82 EurekaUnregisterExpectedMemoryLeak ... 395
83 EurekaUnregisterExpectedMemoryLeakDynArray ... 395
84 EurekaUnregisterExpectedMemoryLeakStr ... 396
85 ExceptionManager ... 396

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

86 ExceptionMessage ... 397
87 ExpandEnvVars ... 398
88 FindDialogClass ... 399
89 FindLowAddress ... 400
90 FindScreenshotRect ... 400
91 FindSender ... 401
92 FreezeNotifyStillAlive ... 401
93 GetActiveFormClass ... 402
94 GetActiveFormText ... 403
95 GetApplicationDescription ... 404
96 GetApplicationName ... 404
97 GetApplicationRect ... 405
98 GetApplicationWnd ... 406
99 GetAsmSize ... 406
100 GetCompanyName ... 407
101 GetCompilationDate ... 408
102 GetComputerName ... 409
103 GetControlClass ... 409
104 GetControlText ... 410
105 GetCurrentModuleCompilationDate ... 411
106 GetCurrentUserPrivileges ... 411
107 GetDiskInfo ... 412
108 GetDisplayDPI ... 412
109 GetDisplayMode ... 413
110 GetEnvVariable ... 413
111 GetEurekaLogLabel ... 414
112 GetEurekaLogLicense ... 415
113 GetEurekaLogLicenseType ... 416
114 GetEurekaLogPath ... 417
115 GetEurekaLogStatus ... 417
116 GetEurekaLogVersion ... 418
117 GetExceptionModuleName ... 419
118 GetExceptionModuleVersion ... 419
119 GetFileIcon ... 420
120 GetFileSize ... 421
121 GetFunctionSize ... 422
122 GetLocationInfo ... 423
123 GetLocationInfoStr ... 423
124 GetMainWnd ... 424
125 GetModuleType ... 425
126 GetMXServerFromEmail ... 426
127 GetParentWnd ... 426

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

128 GetPerformanceCount ... 427
129 GetReadableSize ... 428
130 GetTaskBarWnd ... 429
131 GetTracer ... 430
132 GetTracerClass ... 431
133 GetUserEMail ... 432
134 GUIFont ... 432
135 HandleException ... 433
136 HookWin32API ... 434
137 IfElse ... 435
138 IniReadString ... 435
139 InitCheckFreeze ... 436
140 InitSalt ... 438
141 InitWER ... 440
142 IniWriteString ... 440
143 InstructionToString ... 441
144 IntoIDE ... 442
145 IsAppType ... 442
146 IsBPL ... 443
147 IsCBuilder ... 444
148 IsCGI ... 445
149 IsCGIConsole ... 445
150 IsCGIWin ... 446
151 IsCLX ... 446
152 IsCompiledWithPackages ... 447
153 IsConsole ... 447
154 IsCPL ... 448
155 IsDebugged ... 449
156 IsDLL ... 449
157 IsDynamicRTL ... 450
158 IsEurekaLogActive ... 450
159 IsEurekaLogActive ... 452
160 IsEurekaLogActiveInThread ... 453
161 IsEurekaLogInstalled ... 454
162 IsEXE ... 455
163 IsExpected ... 455
164 IsFMX ... 456
165 IsFreezeCheckEnabled ... 457
166 IsFreezeCheckInstalled ... 458
167 IsGUI ... 458
168 IsIntraWeb ... 459
169 IsISAPI ... 459

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

170 IsLLVM ... 460
171 IsMemLeaksEnabled ... 461
172 IsMemLeaksInstalled ... 461
173 IsModuleLibrary ... 462
174 IsReadableFile ... 463
175 IsRTLCompiledWithStackFrames ... 463
176 IsService ... 464
177 IsStackFramesProc ... 464
178 IsValidBlockAddr ... 465
179 IsValidClass ... 466
180 IsValidCodeAddr ... 467
181 IsValidHandle ... 468
182 IsValidObject ... 468
183 IsVCL ... 469
184 IsVirtualized ... 470
185 IsWeb ... 470
186 IsWin10 ... 471
187 IsWin11 ... 472
188 IsWin64 ... 472
189 IsWin7 ... 473
190 IsWin8 ... 474
191 IsWin81 ... 474
192 IsWinVista ... 475
193 IsWinXP ... 476
194 IsWorkstation ... 476
195 IsWritableFile ... 477
196 LastBugID ... 478
197 LocationToStr ... 478
198 MainThreadTopOfStack ... 479
199 MarkExpected ... 480
200 MD5Hash ... 480
201 MD5HMAC ... 481
202 MemLeaksGetName ... 482
203 MemLeaksName ... 483
204 MemLeaksNameDynArray ... 484
205 MemLeaksNameStr ... 485
206 MemLeaksOwn ... 485
207 MemLeaksOwnDynArrayBoth ... 486
208 MemLeaksOwnDynArrayChild ... 487
209 MemLeaksOwnDynArrayParent ... 487
210 MemLeaksOwnStr ... 488
211 MemLeaksSetParentBlock ... 489

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

212 MemLeaksSetParentBlockDynArrayBoth ... 489
213 MemLeaksSetParentBlockDynArrayChild ... 489
214 MemLeaksSetParentBlockDynArrayParent ... 490
215 MemLeaksSetParentBlockStr ... 491
216 MemProtect ... 491
217 MemUnprotect ... 493
218 MemWipeStack ... 494
219 MonoFont ... 495
220 NameThread ... 496
221 NameThreadForDebugging ... 496
222 PageSize ... 497
223 PauseFreezeCheck ... 497
224 PrimaryInstance ... 498
225 RaiseExpected ... 499
226 RaiseFreezeException ... 499
227 RaiseOuterException ... 500
228 RaiseOuterExceptionExpected ... 501
229 RegisterDialogClass ... 502
230 RegisterDialogClassFirst ... 503
231 RegisterEventAttachedFilesRequest ... 504
232 RegisterEventBeginReportGen ... 505
233 RegisterEventCustomBugID ... 506
234 RegisterEventCustomButtonClick ... 507
235 RegisterEventCustomDataRequest ... 508
236 RegisterEventCustomFileName ... 509
237 RegisterEventCustomWebFieldsRequest ... 510
238 RegisterEventEndReportGen ... 511
239 RegisterEventExceptionAction ... 512
240 RegisterEventExceptionError ... 513
241 RegisterEventExceptionNotify ... 514
242 RegisterEventHelpButtonClick ... 515
243 RegisterEventPasswordRequest ... 516
244 RegisterEventRaise ... 517
245 RegisterEventReproduceRequest ... 518
246 RegisterEventZippedFilesRequest ... 519
247 RegisterSender ... 520
248 RegisterSenderFirst ... 520
249 RegKeyCheck ... 521
250 RegKeyRead ... 522
251 RegKeyWrite ... 523
252 ResumeFreezeCheck ... 524
253 RSADecrypt ... 524

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

254 RSADestroyKey ... 527
255 RSAEncrypt ... 528
256 RSAGenKey ... 531
257 RSALoadPrivateKey ... 532
258 RSALoadPublicKey ... 533
259 RSASavePrivateKey ... 534
260 RSASavePublicKey ... 535
261 RSASign ... 536
262 RSAVerify ... 538
263 SafeExec ... 540
264 Screenshot ... 541
265 ScreenshotFree ... 543
266 ScreenshotInit ... 543
267 ScreenshotSave ... 544
268 SDBMHash ... 545
269 SecureFree ... 545
270 SecureFree ... 547
271 SetAppType ... 547
272 SetEurekaLogState ... 548
273 SetEurekaLogState ... 549
274 SetEurekaLogStateInThread ... 550
275 SetExceptionMessage ... 551
276 SetupConfig ... 552
277 SetUserEmail ... 553
278 SHA1Hash ... 553
279 SHA1HMAC ... 554
280 SHA256Hash ... 555
281 SHA256HMAC ... 556
282 ShowExpected ... 557
283 SignatureGetInfo ... 558
284 TEADecrypt ... 559
285 TEADefaultKey ... 563
286 TEADeriveKey ... 565
287 TEADeriveKeyHMAC ... 569
288 TEADestroyKey ... 572
289 TEAEncrypt ... 572
290 TEAInitSessionKey ... 576
291 ThreadData ... 577
292 ThreadDataEnum ... 578
293 TwofishDecrypt ... 578
294 TwofishDeriveKey ... 580
295 TwofishDeriveKeyHMAC ... 582

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

296 TwofishDestroyKey ... 583
297 TwofishEncrypt ... 583
298 TwofishInitIV ... 585
299 TwofishInitKey ... 586
300 TZipCompressionToString ... 586
301 UnRegisterEventAttachedFilesRequest ... 587
302 UnRegisterEventBeginReportGen ... 588
303 UnRegisterEventCustomBugID ... 589
304 UnRegisterEventCustomButtonClick ... 589
305 UnRegisterEventCustomDataRequest ... 590
306 UnRegisterEventCustomFileName ... 591
307 UnRegisterEventCustomWebFieldsRequest ... 592
308 UnRegisterEventEndReportGen ... 593
309 UnRegisterEventExceptionAction ... 594
310 UnRegisterEventExceptionError ... 594
311 UnRegisterEventExceptionNotify ... 595
312 UnRegisterEventHelpButtonClick ... 596
313 UnRegisterEventPasswordRequest ... 597
314 UnRegisterEventRaise ... 598
315 UnRegisterEventReproduceRequest ... 598
316 UnRegisterEventZippedFilesRequest ... 599
317 WCTDone ... 600
318 WCTInit ... 600

Part XI All Types 603

1 PCustomName ... 607
2 PELLocationInfo ... 608
3 PInstruction ... 608
4 PPointerRecord ... 608
5 PSalt ... 609
6 PtrUInt ... 609
7 PTwofishInitVector ... 610
8 TAppType ... 610
9 TBaseDialogClass ... 611
10 TBlockType ... 611
btCppObject ... 612
btData ... 612
btDynArray ... 612
btObject ... 613
btString ... 613
11 TBugID ... 613
12 TCertificateInfo ... 613
Encryption ... 614
Hash ... 614
Issuer ... 615

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IssuerID ... 615
NotAfter ... 616
NotBefore ... 617
Serial ... 617
Subject ... 618
SubjectID ... 618
Version ... 619
13 TCheckType ... 619
ctEqual ... 620
ctGetLastError ... 620
ctHResult ... 620
ctNone ... 621
ctNotEqual ... 621
ctNTStatus ... 621
14 TCRC16Hash ... 621
15 TCRC32Hash ... 622
16 TCustomName ... 622
17 TDumpCompareFunc ... 622
18 TDumpFilterFunc ... 623
19 TELEvAttachedFilesRequestMeth ... 624
20 TELEvAttachedFilesRequestProc ... 625
21 TELEvCustomBugIDMeth ... 625
22 TELEvCustomBugIDProc ... 626
23 TELEvCustomButtonClickMeth ... 627
24 TELEvCustomButtonClickProc ... 628
25 TELEvCustomDataRequestMeth ... 629
26 TELEvCustomDataRequestProc ... 631
27 TELEvCustomFileNameMeth ... 632
28 TELEvCustomFileNameProc ... 633
29 TELEvCustomWebFieldsRequestMeth ... 634
30 TELEvCustomWebFieldsRequestProc ... 635
31 TELEvExceptionActionMeth ... 636
32 TELEvExceptionActionProc ... 638
33 TELEvExceptionErrorMeth ... 640
34 TELEvExceptionErrorProc ... 642
35 TELEvExceptionNotifyMeth ... 643
36 TELEvExceptionNotifyProc ... 645
37 TELEvHelpButtonClickMeth ... 647
38 TELEvHelpButtonClickProc ... 648
39 TELEvPasswordRequestMeth ... 649
40 TELEvPasswordRequestProc ... 651
41 TELEvRaiseMeth ... 652
42 TELEvRaiseProc ... 653
43 TELEvReportGenMeth ... 655
44 TELEvReportGenMeth ... 656
45 TELEvReportGenProc ... 657

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

46 TELEvReportGenProc ... 657
47 TELEvReproduceRequestMeth ... 658
48 TELEvReproduceRequestProc ... 659
49 TELEvZippedFilesRequestMeth ... 660
50 TELEvZippedFilesRequestProc ... 661
51 TELLocationInfo ... 662
Address ... 663
BinaryFileName ... 663
ClassName ... 663
DebugDetail ... 664
DebugInfo ... 664
LineNumber ... 665
Module ... 665
ModuleName ... 666
ModuleType ... 666
Offset ... 666
OffsetFromLineNumber ... 667
OffsetFromProcName ... 667
ProcedureName ... 668
ProcOffsetLine ... 668
SourceName ... 669
UnitName ... 669
52 TELUniversalSenderClass ... 669
53 TELUseParent ... 670
eupChild ... 671
eupNone ... 671
eupParent ... 671
54 TEncryptBuffer ... 671
55 TEurekaActionType ... 672
atCollectedExceptionInfo ... 674
atCollectingExceptionInfo ... 675
atEurekaLogInitialized ... 675
atEurekaLogInitializing ... 675
atEventHandler ... 675
atExceptionHandled ... 675
atExceptionHandling ... 676
atGeneral ... 676
atRestarting ... 676
atSavedLogFile ... 676
atSavingLogFile ... 676
atSendCancelled ... 676
atSending ... 677
atSent ... 677
atShowedExceptionInfo ... 677
atShowingExceptionInfo ... 677
atTerminating ... 677
56 TEurekaCallStack ... 678
57 TEurekaCallStackClass ... 678
58 TEurekaDebugDetail ... 678
ddModule ... 679
ddNone ... 679
ddProcedure ... 679
ddSourceCode ... 680

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ddUnit ... 680
59 TEurekaDebugDetails ... 680
60 TEurekaLogFileType ... 680
ftBugReport ... 682
ftHTMLPage ... 682
ftLog ... 682
ftMiniDump ... 682
ftSavedBugReport ... 682
ftScreenshot ... 683
ftXML ... 683
ftZIP ... 683
ftZIPCopy ... 683
61 TEurekaModuleType ... 683
62 TEurekaStackList ... 684
63 TExceptionNotifyMode ... 684
emD2009 ... 685
emFull ... 686
emHook ... 686
emNull ... 686
emUnknown ... 686
64 TFreezeThreadClass ... 686
65 TInstruction ... 687
Accessible ... 687
Next ... 688
PlatformSpecific ... 688
Start ... 689
Valid ... 689
66 TIssueIDType ... 690
67 TLogMsgContent ... 691
68 TLogMsgType ... 691
69 TLogType ... 692
elmcBitmap ... 695
elmcCollection ... 695
elmcColor ... 695
elmcComponentAsText ... 696
elmcCustomHierarchy ... 696
elmcCustomProperties ... 696
elmcCustomTable ... 696
elmcCustomText ... 696
elmcMemory ... 697
elmcMemoryStatus ... 697
elmcObject ... 697
elmcRegistry ... 697
elmcStackTrace ... 697
elmcStringList ... 697
elmcSystemInfo ... 698
elmcText ... 698
elmcTextFile ... 698
elmcVersionInfo ... 698
elmcWindow ... 698
elmcXmlData ... 699
elmtAppStartSeparator ... 699
elmtBlue ... 699
elmtCheckPoint ... 699

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

elmtClear ... 699
elmtCollapsedMethod ... 699
elmtEnterMethod ... 700
elmtError ... 700
elmtException ... 700
elmtExitMethod ... 700
elmtExitMethodCollapse ... 700
elmtGreen ... 701
elmtInactiveSeparator ... 701
elmtIndigo ... 701
elmtInfo ... 701
elmtLevel1 ... 701
elmtLevel2 ... 701
elmtLevel3 ... 702
elmtLevel4 ... 702
elmtLevel5 ... 702
elmtLevel6 ... 702
elmtLevel7 ... 702
elmtNote ... 703
elmtOrange ... 703
elmtRed ... 703
elmtReminder ... 703
elmtResetSeparator ... 703
elmtSaveLogFile ... 703
elmtScratchPad ... 704
elmtSeparator ... 704
elmtTiming ... 704
elmtViolet ... 704
elmtWarning ... 704
elmtYellow ... 705

70 TMD5Hash ... 705

71 TMemoryError ... 705

leHang ... 706
leInvalidBlock ... 707
leMultiFree ... 707
leNone ... 707
leObjectInUse ... 707
leOutOfMemory ... 707
leOverflow ... 708
leUninstalled ... 708
leUseAfterFree ... 708

72 TMemoryErrors ... 708

73 TModuleType ... 709

mtLibrary ... 709
mtPackage ... 710
mtProgram ... 710
mtUnknown ... 710

74 TPointerRecord ... 710

BlockState ... 711
BlockType ... 711
CallStack ... 712
SingleSize ... 712

75 TResponse ... 713

ErrorCode ... 713
ErrorMessage ... 714
IssueID ... 714

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IssueIDStr ... 715
IssueIDType ... 715
SendResult ... 716
SuccessMessage ... 716
76 TRSAExport ... 717
rsBLOB ... 718
rsDER ... 718
rsPEM ... 718
77 TRSAImport ... 718
78 TRSAKey ... 719
PrivateKey ... 719
PublicKey ... 720
79 TRSASignature ... 720
80 TSalt ... 721
81 TScreeenshotHandle ... 721
82 TSDBMHash ... 722
83 TSendResult ... 722
srBugClosed ... 724
srBugReopen ... 725
srCancelled ... 725
srConnectionError ... 725
srInvalidInsert ... 725
srInvalidLogin ... 725
srInvalidModify ... 726
srInvalidRead ... 726
srInvalidSearch ... 726
srInvalidSelection ... 726
srNoExceptionInfo ... 726
srObsoleteVersion ... 726
srRestart ... 727
srSent ... 727
srUnknownError ... 727
srUnsupportedSender ... 727
84 TSendScreenshot ... 727
ssActiveWindow ... 729
ssApplication ... 729
ssDesktop ... 729
ssForegroundWindow ... 729
ssNone ... 729
ssPrimary ... 730
ssWorkarea ... 730
85 TSHA1Hash ... 730
86 TSHA256Hash ... 730
87 TSignatureInfo ... 731
Certificates ... 731
FileName ... 732
MoreInfoLink ... 732
ProgramName ... 733
PublisherLink ... 733
Signed ... 734
Timestamp ... 735
TimestampCertificates ... 735
Timestamped ... 736

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

88 TStreamConstructor ... 736
89 TSysErrorCode ... 737
90 TTEAKey ... 737
91 TThreadKind ... 738
tkBeginThread ... 738
tkBeginThreadEx ... 739
tkGeneric ... 739
tkTThread ... 739
tkTThreadEx ... 739
92 TTwofishInitVector ... 739
93 TTwofishKey ... 740
94 TTwofishRAWKey ... 740
95 TZipCompression ... 741
zcBZIP2 ... 743
zcDeflate ... 743
zcDeflate64 ... 743
zcImplode ... 743
zcLZ77 ... 743
zcLZMA ... 744
zcPKImplode ... 744
zcPPMdI1 ... 744
zcReduce1 ... 744
zcReduce2 ... 744
zcReduce3 ... 745
zcReduce4 ... 745
zcReserved11 ... 745
zcReserved13 ... 745
zcReserved15 ... 745
zcReserved16 ... 746
zcReserved17 ... 746
zcShrunk ... 746
zcStored ... 746
zcTERSE ... 746
zcTokenize ... 746
zcWavePack ... 747
96 TZipEndOfCentralHeader ... 747
97 TZipHeader ... 747
HostOS ... 748
MadeByVersion ... 749
RequiredVersion ... 749

Part XII All Globals 753

1 DefaultTracerMask ... 753
2 EEurekaLogCurrentVersion ... 754
3 EEurekaLogLicense ... 754
4 EEurekaLogVersion ... 755
5 EurekaCallStackClass ... 755
6 FreezeThreadClass ... 756
7 LogBuilderClass ... 757
8 CENTRALHEADER ... 757
9 DATADESCRIPTOR ... 758

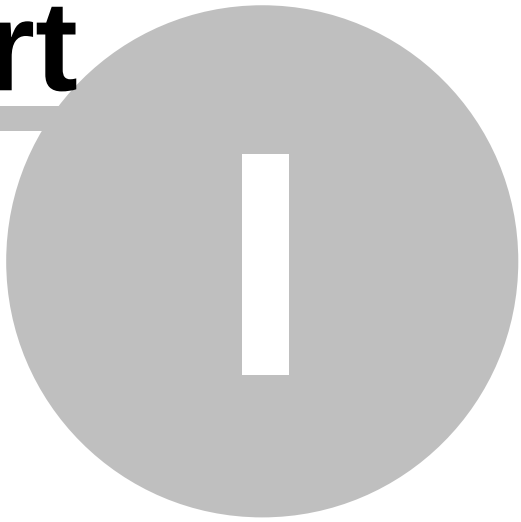
© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10 LOCALHEADER 758
11 ZIPENDOFHEADER 759

Index 760

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part



1 Main routines

IsEurekaLogInstalled 2	Indicates if EurekaLog was included in application
IsEurekaLogActive 3	Returns global EurekaLog's activation status
IsEurekaLogActiveInThread 5	Returns per-thread EurekaLog's activation status
SetEurekaLogActive 6	Changes global EurekaLog activation status
SetEurekaLogActiveInThread 7	Changes per-thread EurekaLog activation status
CurrentEurekaLogOptions 8	Returns EurekaLog's options for current executable
LastBugID 9	Returns BugID of last exception in current thread

See also

- Events 11
- Exception manager 396
- Exception information 180
- Call stack functions 59
- Location routines 68
- Informational functions 134
- Formatting functions 99
- Module functions 114
- Thread functions 136
- Screenshot functions 119
- More functions 303

1.1 IsEurekaLogInstalled

Indicates if EurekaLog is installed in this application.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function IsEurekaLogInstalled: Boolean;
```

Return value

True if EurekaLog is installed, False - otherwise

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

EurekaLog is considered to be installed if:
1. its source code was included into application (see ExceptionLog7 unit 141);
2. executable was processed by EurekaLog expert or command-line compiler (ECC32/EMAKE);
I.e. EurekaLog is installed if both true: source code is included **AND** options is included.

Debug information (which is required to build human-readable call stack) can be included or not - this doesn't matter for installation. EurekaLog can function without debug information.

Alternative to this function can be using of conditional directives.

EurekaLog defines the following conditional symbols:
- EUREKALOG
- EUREKALOG7

Note
This is a wrapper for EBase.IsEurekaLogInstalled function.

Examples

Code (Delphi)

```
uses
   EBase;
// ...
if IsEurekaLogInstalled then
begin
   // Do something if EurekaLog is installed
end;
```

Code (Delphi)

```
{$IFDEF EUREKALOG}
   // Do something if EurekaLog is installed
{$ENDIF}
```

See also

- IsEurekaLogActive 3
- _InstallEurekaLog

1.2 IsEurekaLogActive

Indicates if EurekaLog is active or disabled in this application.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function IsEurekaLogActive: Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

True if EurekaLog is active, False - otherwise.

Remarks

EurekaLog is considered to be active if EurekaLog should do its work. You can control EurekaLog activation by calling SetEurekaLogState 6 procedure.

If EurekaLog was not installed - then it's always disabled (not active).

This is a global status for all application. If you want to check per-thread activation status - use IsEurekaLogActiveInThread 5 function.

If you want to change activation status (enable or disable EurekaLog) - use SetEurekaLogState 6 for global status and SetEurekaLogStateInThread 7 for per-thread status.

By default, EurekaLog is active at startup (if it was installed).

Caution!

This function doesn't respect per-thread activation status. Thus, if EurekaLog was disabled in one thread only - this function will still returns `True`.

This function respects "installed" status, so there is no need for additional call to IsEurekaLogInstalled function.

Note

This function is for the clients only. EurekaLog's code should use _IsEurekaLogActive instead.

Note

This is a wrapper for IsEurekaLogActive 452 EBase.IsEurekaLogActive

Examples

Code (Delphi)

```
if IsEurekaLogActive then
   // this exception will be handled by EurekaLog, since EurekaLog is er
   raise Exception.Create('Error message')
else
   // this exception will not be handled by EurekaLog, since EurekaLog i
   raise Exception.Create('Error message');
```

See also

- IsEurekaLogActiveInThread 5
- IsEurekaLogInstalled 2
- SetEurekaLogState 6
- SetEurekaLogStateInThread 7
- _IsEurekaLogActive
- _InstallEurekaLog

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

1.3 IsEurekaLogActiveInThread

Indicates if EurekaLog is active or disabled in the specified thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function IsEurekaLogActiveInThread(
   const AThreadID: Cardinal = 0
): Boolean;
```

Parameters

AThreadID [in, optional]

TID - thread ID which EurekaLog's status should be checked. Use 0 (default) for current thread.

Return value

True if EurekaLog is active in the specified thread, False - otherwise.

Remarks

This is per-thread version of IsEurekaLogActive 3 function.

This is a per-thread status. If you want to check global activation status - use IsEurekaLogActive 3 function.

If you want to change activation status (enable or disable EurekaLog) - use SetEurekaLogState 6 for global status and SetEurekaLogStateInThread 7 for per-thread status.

Caution!

This function doesn't respect global activation status.

Note

This is a wrapper for EThreadsManager.IsEurekaLogActiveInThread function.

Examples

Code (Delphi)

```
if IsEurekaLogActiveInThread then
   // this exception will be handled by EurekaLog, since EurekaLog is er
   raise Exception.Create('Error message')
else
   // this exception will not be handled by EurekaLog, since EurekaLog i
   raise Exception.Create('Error message');
```

See also

- IsEurekaLogActive 3
- IsEurekaLogInstalled 2
- SetEurekaLogState 6

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- SetEurekaLogStateInThread 7
- _InstallEurekaLog
- _IsEurekaLogActive

1.4 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function SetEurekaLogState(
   const AActivate: Boolean
): Boolean;
```

Parameters

AActivate [in]

`True` - if you want to enable EurekaLog, `False` - if you want to disable EurekaLog.

Return value

Previous EurekaLog state: True if EurekaLog was active, False - otherwise.

Remarks

You can use this procedure to manually enable and disable EurekaLog in your application.

EurekaLog is considered to be active if it should do its work. You can get EurekaLog activation status by calling IsEurekaLogActive 3 procedure.

If EurekaLog is not installed - then it's always disabled (not active).

This function changes global status for all application. If you want to change activation status on per-thread basis - use SetEurekaLogStateInThread 7 procedure.

Note

This is a wrapper for EBase.SetEurekaLogState 549 function.

Examples

Code (Delphi)

```
// Enable EurekaLog in application (by default)
SetEurekaLogState(True);
raise Exception.Create('Error message'); // <- this exception will be ha
```

Code (Delphi)

```
// Disable EurekaLog in application
SetEurekaLogState(False);
raise Exception.Create('Error message'); // <- this exception will not b
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- IsEurekaLogActive 3
- SetEurekaLogStateInThread 7

1.5 SetEurekaLogStateInThread

Enables or disables EurekaLog in the specified thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function SetEurekaLogStateInThread(
   const AThreadID: Cardinal;
   const AActivate: Boolean
): Boolean;
```

Parameters

AThreadID [in, optional]

TID - thread ID which EurekaLog's status should be checked. Use 0 (default) for current thread.

AActivate [in]

`True` - if you want to enable EurekaLog, `False` - if you want to disable EurekaLog.

Return value

Previous EurekaLog state: True if EurekaLog was active, False - otherwise.

Remarks

You can use this procedure to manually enable and disable EurekaLog in the specified thread.

EurekaLog is considered to be active if it should do its work. You can get per-thread EurekaLog activation status by calling IsEurekaLogActiveInThread 5 procedure.

This function changes per-thread activation status. If you want to change global activation status - use SetEurekaLogState 6 procedure.

Note

This is a wrapper for EThreadsManager.SetEurekaLogStateInThread function.

Examples

Code (Delphi)

```
// Enable EurekaLog in current thread (by default)
SetEurekaLogStateInThread(True);
raise Exception.Create('Error message'); // <- this exception will be ha
```

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
// Disable EurekaLog in current thread
SetEurekaLogStateInThread(False);
raise Exception.Create('Error message'); // <- this exception will not b
```

See also

- IsEurekaLogActive 3
- SetEurekaLogState 6

1.6 CurrentEurekaLogOptions

Returns current EurekaLog's options.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function CurrentEurekaLogOptions: TEurekaModuleOptions 216;
```

Return value

Current EurekaLog's options.

Remarks

This function returns options for EurekaLog in current executable module. Options are present if EurekaLog was installed.

Important!

If EurekaLog is not active - any attempt to call this function will result in error message "*Cannot use ''CurrentEurekaLogOptions'' function in module "XYZ" without activate EurekaLog*" and immediate process termination.

This behaviour is "as designed". It's done to avoid cases when you ship your application with EurekaLog's code but without EurekaLog active.

Default options are set based on EurekaLog's projects settings - as set by you in IDE expert or standalone Settings Editor Tool. Options can be changed at run-time at any time.

Note

Some options are read-only. That's because these options can not be changed after activation.

`CurrentEurekaLogOptions` represents global EurekaLog's options. These options can also be captured locally. In this case changes to global options will have no effect for captured options.

For example, when exception is raised - it creates a copy of options. Thus, if you change options globally - this will have no effect on already active exceptions.

If you want to change options for such exceptions - you should use local copy of options (i.e. `Options` property for dialogs, `Options` argument in event handlers, etc.).

See TEurekaModuleOptions 216 for detailed description of available EurekaLog's options.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note

It's highly recommended to use enviroment variables when possible instead of customizing options at run-time. That's because some exceptions can occur before you get chance to setup options.

Note

This is a wrapper for EModules.CurrentEurekaLogOptions function.

Examples

Code (Delphi)

```
// Disable sending if run on my machine
if ApplicationRunOnDeveloperMachine then
   CurrentEurekaLogOptions.SenderClasses := '';
```

1.7 LastBugID

Returns BugID of last exception in current thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function LastBugID: TBugID 613;
```

Return value

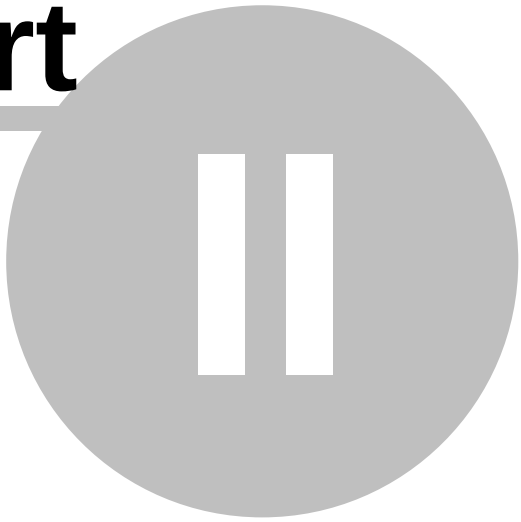
BugID of last thread exception or 0 if there was no exception raised.

See also

- BugID

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part



2 Events

OnRaise 12	Fires on each exception raising
OnExceptionNotify 14	Fires on each exception handling
OnExceptionAction 16	Fires for exception processing stages; allows you to fine-control exception handling
OnExceptionError 18	Fires on any error during exception processing
OnPasswordRequest 19	Use this event to supply a password to *decrypt* call stacks
OnAttachedFilesRequest 21	Include additional files into bug report (separate files)
OnZippedFilesRequest 22	Include additional files into zipped bug report
OnReproduceRequest 23	Supply a custom steps-to-reproduce data
OnCustomFileName 24	Allows you to alter any file name
OnCustomBugID 25	Customize BugID by supplying sub-ID or replace it completely with your own values
OnCustomDataRequest 26	Include additional data into bug report (text)
OnCustomWebFieldRequest 27	Customize request to web server or bug tracker with additional custom data
OnCustomButtonClick 28	Fires when "Custom" button is clicked in error dialog
OnHelpButtonClick 29	Fires when "Help" button is clicked in error dialog
OnBeginReportGen/OnEndReportGen 30	Fires before and after bug report generating

Warning: Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

A typical order of events may go like this:

- Raise stage:
  - OnExceptionAction atCollectingExceptionInfo
  - OnRaise
  - OnExceptionAction atCollectedExceptionInfo
  - **Debugger notification about exception**
- Handle stage:
  - OnExceptionNotify
  - OnCustomBugID
  - OnCustomFileName ftSavedBugReport
  - OnCustomFileName ftBugReport
  - OnCustomFileName ftZIP
  - OnCustomFileName ftZIPCopy
  - OnCustomFileName ftXML
  - OnCustomFileName ftScreenshot
  - OnCustomFileName ftHTMLPage
  - OnCustomFileName ftMiniDump
  - OnCustomDataRequest
  - OnExceptionAction atSavingLogFile
  - OnBeginReportGen
  - OnEndReportGen
  - OnExceptionAction atSavedLogFile
  - OnExceptionAction atShowingExceptionInfo
  - OnReproduceRequest
  - OnExceptionAction atShowedExceptionInfo
  - OnExceptionAction atSavingLogFile
  - OnExceptionAction atSavedLogFile
  - OnZippedFilesRequest
  - OnAttachedFilesRequest
  - OnExceptionAction atSending
  - OnCustomWebFieldsRequest
  - OnExceptionAction atSent
  - OnExceptionAction atRestarting

See also

- Events description
- TEurekaLogEvents component 33
- Functions 2

2.1 OnRaise

Method event handler for OnRaise event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvRaiseMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ACallNextHandler: Boolean
) of object;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

OnRaise event is fired for each raised exception. You can use this event to collect additional information about exception, to change existing information or place any other custom logic.

Important!

Do not place exception processing code into event handler for this event. Use OnExceptionNotify 141 event instead.

Note

OnRaise event is invoked when EurekaLog detects that a new exception is raised. Depending on project configuration and environment - it can happen before or after an exception is actually thrown by CPU. This means that not all exception information may be collected. For example, a call stack may be `nil` - because building call stack for the exception requires CPU context record - which is not available until exception is actually thrown by CPU.

Note

OnRaise event is wrapped into OnActionNotify events with atCollectingExceptionInfo/ atCollectedExceptionInfo.

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
raise Exception.Create('Error'); // - event will be raised here
```

See this article 83 for example of registering event handlers.

Sample event handler may look like this:

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
procedure TMyObject.OnRaise(AExceptionInfo: TEurekaExceptionInfo; var AC
begin
   // Collect additional information for exception
   AExceptionInfo.AdditionalInfo := 'Sample additional data';
end;
```

See also

- Event handler registration 517
- Procedural OnRaise event handler 653
- EurekaLog component 33
- General events concept 83
- Other events 11

2.2 OnExceptionNotify

Method event handler for OnExceptionNotify event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionNotifyMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AHandle: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AHandle [out]
`True` - (default) process exception with EurekaLog, `False` - process exception with standard exception handler

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

> Tip
>
> Set `ACallNextToHandler` to `False`, if you set `AHandle` to `False`.

Remarks

By default: OnExceptionNotify event is fired for any exception which wasn't handled by the code.

> Note
>
> If "Catch handled exception" option is enabled - this event will also fire for any handled exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

You can determinate the kind of exception (handled or unhandled) by analyzing properties of `AExceptionInfo` argument (see Handled property).

You can use this event to disable processing of certain exceptions by EurekaLog.

Tip

You can also use exception filters instead of this event.

Use OnExceptionAction 638 event to fine-control processing of exceptions.

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
finally
    ...
end;
   // - OnExceptionNotify will be called
```

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
except
    ...
end;
   // - OnExceptionNotify will not be called, since exception was handle
```

See this article 83 for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure MyExceptionNotify(const ACustom: Pointer; AExceptionInfo: TEur
begin
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    // Disable exception processing by EurekaLog
    Handle := False;
    ACallNextHandler := False;
end;
```

See also

- Event handler registration 514
- Procedural OnExceptionNotify event handler 645
- EurekaLog component 33
- General events concept 83
- Other events 11

2.3 OnExceptionAction

Method event handler for OnExceptionAction event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionActionMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AEurekaAction: TEurekaActionType 672;
   const AAdditionalInfo: String;
   var AExecute: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

AEurekaAction [in]
Action to perform.

AAdditionalInfo [in, optional]
Additional information about action (see remarks).

AExecute [out]
"*Before-like*" actions: `True` - (default) perform action, `False` - cancel/skip action;

"*After-like*" actions: `True` - action is successful, `False` - action was unsuccessful;

> Note
>
> This argument is ignored for atSendCancelled 676.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

> Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Set to `False`, if you set `AExecute` to `False`.

Remarks

OnExceptionAction event is fired for any action with exception during exception processing: saving to bug report, sending via Internet, etc. You can use this event to alter actions, cancel/skip them, etc.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 677 and atSent 677 actions. For these actions - `AAdditionalInfo` argument holds send engine class name. This information is useful if you use several send methods (for example, HTTP and e-mail). See also 41.

For "before"-like actions: `AExecute` will determinate if EurekaLog should execute the action. You can use this to skip certain actions (e.g. skip sending, skip dialog, skip saving bug report, etc.).

For "after"-like actions: `AExecute` will determinate if EurekaLog should consider action as succeeded (or failed). You can use this to skip errors.

Note

Basically, for most actions EurekaLog follow such pattern (pseudo-code):

Code (Delphi)

```
repeat
          MakeCall := DoEventExceptionAction(... "before" ...);
          try
          if MakeCall then
          Result := PerformAction
          else
          Result := Cancelled;
          except
          Result := UnknownError;
          end;
          Result := DoEventExceptionAction(... "after" ...);

          if Failed(Result) then
          Retry := DoEventExceptionError(...)
          else
          Retry := False;
         until not Retry;
```

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event handler will be c
except
   Application.HandleException();
       // - OnExceptionNotify will be called
      // - set of OnExceptionAction events will be called
       // (one event for each action (saving, sending, etc).)
   end;
```

See also

- Event handler registration 512
- Procedural OnExceptionAction event handler 638
- EurekaLog component 33
- General events concept 83
- Other events 11

2.4 OnExceptionError

Method event handler for OnExceptionError event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionErrorMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AEurekaAction: TEurekaActionType 672;
   const AAdditionalInfo: String;
   var ARetry: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

AEurekaAction [in]
Failed action.

AAdditionalInfo [in]
Reason why action has failed.

ARetry [out]
`True` - repeat failed action, `False` - (default) fail operation.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Tip
Set to `False`, if you set `ARetry` to `True`.

Remarks

OnExceptionError event is fired for any failed operation during exception processing: saving to bug report, sending via Internet, etc. You can use this event to correct settings and retry the operation or to use a fallback fail code, etc.

`AExceptionInfo` represents original exception which is being processed. It may be `nil`.

`AAdditionalInfo` represents secondary error which failed `AEurekaAction` step.

Tip
Be sure to do something before switching `ARetry` to `True`. Repeating failed action without any attempt to correct settings / environment will simply fail again.

Warning
Be careful when setting `ARetry` to `True`: the failed action will be repeated and may raise the same error again. Ensure you have some sort of check to avoid infinite loop.

Caution!
Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 513
- Procedural OnExceptionError event handler 642
- EurekaLog component 33
- General events concept 83
- Other events 11

2.5 OnPasswordRequest

Method event handler for OnPasswordRequest event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvPasswordRequestMeth = procedure(
   const AModule: HMODULE;
   const APID: PGUID;
   const APasswordHash: Pointer;
   var APassword: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AModule [in]
Module handle which password is retrieved.

APID [in, optional]
Pointer to unique project ID. Can be used to pick password from cache. This parameter is always assigned for EurekaLog 7+ modules.

APasswordHash [in, optional]
Pointer to EHash.TSHA1Hash 730. SHA1 hash of the UTF8-encoded password. Can be used to validate password. This parameter is always assigned for EurekaLog 7+ modules.

APassword [out]
Password for debug information.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

EurekaLog includes a OnPasswordRequest event for interaction with the user program when a raised exception is handled by EurekaLog and debug information was saved as encrypted. This event allow EurekaLog to obtain the exact password required to can decrypt all the encrypted data, otherwise all the data will be store in the bug report file in encrypted form, also error dialogs will show bug report in encrypted form.

The developer of application may use EurekaLog Viewer tool to decrypt encrypted bug reports.

You can use this event to supply your password to EurekaLog. By default this password is not stored in your executables. It's impossible to decrypt encrypted debug information without a password.

You can either ask user for password at run-time or retrieve it from application's configuration.

Note

This event will not be called if password was obtained by some other means (for example, loaded from local cache, or supplied via `--el_password=` cmd line argument, etc.).

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 516
- Method OnPasswordRequest event handler 651
- EurekaLog component 33
- General events concept 83
- Other events 11

2.6 OnAttachedFilesRequest

Method event handler for OnAttachedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

`AExceptionInfo` [in]
Information about exception.

`ATempFolder` [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

`AAttachedFiles` [out]
List of attached file names. Use full absolute paths.

`ACallNextHandler` [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as separate files.

To include files into bug report file - use OnZippedFilesRequest event.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 504
- Procedural OnAttachedFilesRequest event handler 625
- OnZippedFilesRequest event handler 22
- EurekaLog component 33
- General events concept 83
- Other events 11

2.7 OnZippedFilesRequest

Method event handler for OnZippedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ATempFolder [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

AAttachedFiles [out]
List of attached file names. Use full absolute paths.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as inside packed report (.elp file).

To include files into bug report file - use OnAttachedFilesRequest event.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 519
- Procedural OnZippedFilesRequest event handler 661
- OnAttachedFilesRequest event handler 21
- EurekaLog component 33
- General events concept 83
- Other events 11

2.8 OnReproduceRequest

Method event handler for OnReproduceRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReproduceRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AReproduceText: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AReproduceText [in, out]
Text to be used as "steps to reproduce".

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify "steps to reproduce". A typical example would be appending run-time information to steps to reproduce. Alternatively, you may use OnCustomData event handler.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 518
- Procedural OnReproduceRequest event handler 659

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EurekaLog component 33
- General events concept 83
- Other events 11

2.9 OnCustomFileName

Method event handler for OnCustomFileName event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomFileNameMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AFileType: TEurekaLogFileType 680;
   var AFileName: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AFileType [in]
Kind of file name to modify.

AFileName [in, out]
Actual file name that may be changed. Use full absolute file path.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify file names.

Even though you can relocate file in arbitrary folder - it's highly recommended to keep original folder and extension.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 509
- Procedural OnCustomFileName event handler 633
- EurekaLog component 33
- General events concept 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Other events 11

2.10 OnCustomBugID

Method event handler for OnCustomBugID event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomBugIDMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ABugID, ACustomBugID: Word;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ABugID [in, out]
Primary BugID generated by EurekaLog.

ACustomBugID [out]
Custom BugID.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event handler to override BugID creation.

You can define your own sub-ID by altering `ACustomBugID`. Alternatively you may also replace BugID generated by EurekaLog.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 506
- Procedural OnCustomBugID event handler 626
- EurekaLog component 33
- General events concept 83
- Other events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

2.11 OnCustomDataRequest

Method event handler for OnCustomDataRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomDataRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   ADataFields: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ALogBuilder [in]
Log builder class, representing bug report.

ADataFields [out]
Custom fields to add into bug report (as `name=value` pairs).

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before saving bug report to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `ADataFields.Values['Field-Name'] := 'Field-Value'`. Field name can be arbitrary and are choosen by you.

Warning

Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 508

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Procedural OnCustomDataRequest event handler 631
- EurekaLog component 33
- General events concept 83
- Other events 11

2.12 OnCustomWebFieldsRequest

Method event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomWebFieldsRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ASender: TObject;
   AWebFields: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

ASender [in]
Current sender class (child class of `TELUniversalSender`).

AWebFields [in]
Web fields to send to web-site (as `name=value` pairs).

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before sending to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `AWebFields.Values['Field-Name'] := 'Field-Value'`. Field name depends on used send method.

For example, if you are sending to Mantis - then field name would be custom field name as it appears in Mantis configuration (e.g. like `'cf_count'`).

This event is not called if send method does not need / support custom fields.

Note

`AExceptionInfo` param will be missing if sending is performed outside of exception processing. E.g. if sending is called manually.

Warning

Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 510
- Procedural OnCustomWebFieldsRequest event handler 635
- EurekaLog component 33
- General events concept 83
- Other events 11

2.13 OnCustomButtonClick

Method event handler for OnCustomButtonClick event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACloseDialog: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

`AExceptionInfo` [in]
Information about exception.

`AOptions` [in]
Current EurekaLog options.

`ADialog` [in]
Current dialog class (child class of `TBaseDialog`).

`ACloseDialog` [out]
True - close dialog after return, False (default) - do not close dialog.

`ACallNextHandler` [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked when end-user has clicked on custom button in error dialog. You should put your

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

custom action code there. For example, if you setup custom button to be "check for updates" - then you should put your update checking code there.

Note

There is a similar event 29
- Custom button is shown only when you specify it in options. It's shown for errors (i.e. bug reports).
- Help button is shown automatically whenever there is help context for known exception (which do not generate bug report).

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also
- Event handler registration 507
- Procedural OnCustomButtonClick event handler 628
- OnHelpButtonClick event handler 29
- EurekaLog component 33
- General events concept 83
- Other events 11

2.14 OnHelpButtonClick

Method event handler for OnHelpButtonClick event.

Unit
EEvents 83

Syntax

Code (Delphi)

```
TELEvHelpButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACallNextHandler: Boolean
) of object;
```

Parameters
`AExceptionInfo` [in]
Information about exception.

`AOptions` [in]
Current EurekaLog options.

`ADialog` [in]
Dialog class.

`ACallNextHandler` [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This event is invoked when end-user has clicked on "Help" button in error dialog.

Help button is displayed automatically for known exceptions (i.e. exceptions that are not errors; they do not generate bug reports).

Note
There is a similar event 28.

Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 515
- Procedural OnHelpButtonClick event handler 648
- OnCustomButtonClick event handler 28
- EurekaLog component 33
- General events concept 83
- Other events 11
- RaiseExpected routine 499

2.15 OnBeginReportGen/OnEndReportGen

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReportGenMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

`AExceptionInfo` [in]
Information about exception.

`ALogBuilder` [in]
Log builder class.

`AOptions` [in]
Current EurekaLog options.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is called before creating bug report. You can use this event to intercept creation of bug reports. The same event handler type is used for both OnReportBegtinGen and OnReportEndGen events.

`ALogBuilder` is TBaseLogBuilder 152 instance. You need to use type cast.

Note
A more convenient case of this type 655 is declared in ELogBuilder 101 unit. Use it instead of this type whenever possible. Type from ELogBuilder doesn't require you to use typecast.

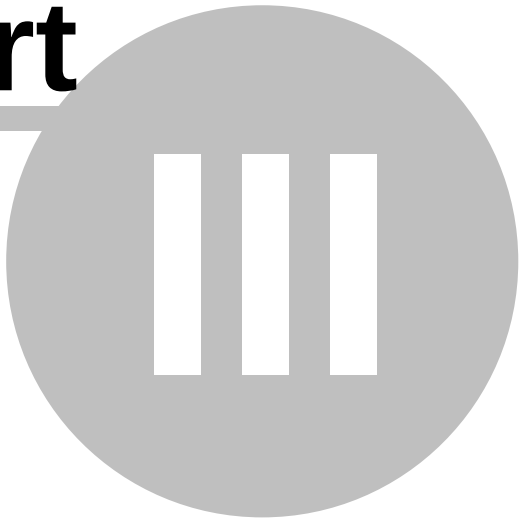
Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration
- Procedural OnReportBeginGen/OnReportEndGen event handler 18
- EurekaLog component 33
- General events concept 83
- Other events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part



3 Component

EurekaLog 7 component.

Unit

EComponent 62

Syntax

Code (Delphi)

```
TEurekaLogEvents = class(TComponent);
```

Remarks

EurekaLog 7 component can be used as simple way to react to EurekaLog's events.

This component provides a quick access to EurekaLog events. Normally a EurekaLog event handler is procedure or method which are registered (subscribed) to events via routines in EEvents 83 unit. By using this component you can just double-click on events in IDE's Object Inspector to create handlers. No code writing is necessary.

Caution!

While using component is very easy, you should be aware that any component lives as long as its container (form, frame or data module) lives. Thus, any event handler which you create with component will be inactive after your form close.

For this reason it's best to avoid component at all and use event handlers directly. Any event handler can be registered via EEvents 83 unit. Best place to register event handler - `initialization` sections of your units.

If you still want to use EurekaLog component - best option is to place it on the main form of your application.

Note

It's perfectly fine to have multiple EurekaLog components in your application.

Events 34	Events
private	Private members
published	Published members

Examples

For sample code of events handlers - please see description of specific events.

See also

- Events index 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

3.1 Events

OnAttachedFilesRequest 210	Wrapper for OnAttachedFilesRequest 21 event.
OnBeginReportGen 211	Wrapper for OnBeginReportGen 30 event.
OnCustomBugID 211	Wrapper for OnCustomBugID 25 event.
OnCustomButtonClick 212	Wrapper for OnCustomButtonClick 28 event.
OnCustomDataRequest 212	Wrapper for OnCustomDataRequest 26 event.
OnCustomFileName 212	Wrapper for OnCustomFileName 24 event.
OnCustomWebFieldsRequest 213	Wrapper for OnCustomWebFieldsRequest 27 event.
OnEndReportGen 213	Wrapper for OnEndReportGen 30 event.
OnExceptionAction 214	Wrapper for OnExceptionAction 16 event.
OnExceptionError 214	Wrapper for OnExceptionError 18 event.
OnExceptionNotify 214	Wrapper for OnExceptionNotify 14 event.
OnPasswordRequest 215	Wrapper for OnPasswordRequest 19 event.
OnRaise 215	Wrapper for OnRaise 12 event.
OnReproduceRequest 215	Wrapper for OnReproduceRequest 23 event.
OnZippedFilesRequest 216	Wrapper for OnZippedFilesRequest 22 event.

See also

- TEurekaLogEvents 33
- EComponent 62

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part

4 Dialogs

Base class 36	Base class for all dialogs
Null dialog 37	Dialog class which does nothing
RTL dialog 37	Dialog class which executes standard exception processing
Console 37	Dialog class for console applications
MessageBox 37	Simple dialog class for GUI applications
MS Classic 39	Main dialog class for GUI applications
EurekaLog 37	Debug dialog class for GUI applications
CGI 37	Dialog class for CGI applications
ISAPI 37	Dialog class for ISAPI applications
IntraWeb 37	Dialog class for IntraWeb applications
WER 37	Dialog class which invokes OS exception processing
System Log 37	Dialog class for NT services

See also

- Dialogs description

4.1 Base class

Base class for all dialogs.

Unit

EDialog 72

Syntax

Code (Delphi)

```
TBaseDialog = class(TObject);
```

Remarks

This is base abstract class for error dialogs in EurekaLog. It provides service methods and implements common interface.

Actual (non abstract) dialogs are implemented as child sub-classes for this class. Each dialog class must be registered via RegisterDialogClass 502 or RegisterDialogClassFirst 503 functions.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Each child class must implement at least ShowModalInternal method (protected), where it must implement functionality of the dialog.

4.2 Null dialog

Enter topic text here.

4.3 RTL dialog

Enter topic text here.

4.4 Console

Enter topic text here.

4.5 MessageBox dialog

Enter topic text here.

4.5.1 MessageBox detailed dialog

Enter topic text here.

4.6 MS Classic

Enter topic text here.

4.7 EurekaLog

Enter topic text here.

4.7.1 EurekaLog detailed

Enter topic text here.

4.8 CGI

Enter topic text here.

4.9 ISAPI

Enter topic text here.

4.10 IntraWeb

Enter topic text here.

4.11 WER

Enter topic text here.

4.12 System Log

Enter topic text here.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part V

5 Debug information providers

EurekaLog 39	Build-in class for EurekaLog format
MAP 39	Provider class for .map files
TD32 39	Provider class for .tds files (Turbo Debugger)
JEDI 39	Provider class for JclDebug format
madExcept 39	Provider class for madExcept format
Microsoft 39	Provider class for Microsoft formats
DLL export table 39	Provider class for DLL function exports

See also

- Providers description

5.1 EurekaLog

Enter topic text here.

5.2 .map files

Enter topic text here.

5.3 Turbo Debugger (TDS)

Enter topic text here.

5.4 JEDI (Jcl)

Enter topic text here.

5.5 madExcept

Enter topic text here.

5.6 Microsoft

Enter topic text here.

5.7 DLL export table

Enter topic text here.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part VI

6 Send engines

mailto 41	Send class for "mailto" protocol (send via Shell)
Simple MAPI 41	Send class for Simple MAPI protocol
MAPI 42	Send class for MAPI protocol
SMTP 42	Send class for SMTP protocol
HTTP(S) 43	Send class for HTTP/HTTPS protocols
FTP 43	Send class for FTP protocol
Mantis 44	Send class for Mantis bug tracker
FogBugz 45	Send class for FogBugz bug tracker
BugZilla 168	Send class for BugZilla bug tracker
JIRA 170	Send class for JIRA bug tracker

See also

- Send engines description

6.1 mailto

Implements sending via mailto: protocol

Unit

ESendMailShell 128

Syntax

Code (Delphi)

```
TELMailShellSender = class(TELMailSender);
```

Remarks

Works via ShellExecute.

6.2 Simple MAPI

Implements sending via Simple MAPI

Unit

ESendMailSMAPI 128

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELMailSimpleMAPISender = class(TELMailSender);
```

Remarks

Works via SMTP stub, can work in cross-bitness.

6.3 MAPI

Implements sending via full MAPI protocol.

Unit

ESendMailMAPI 127

Syntax

```
Code (Delphi)
TELMailMAPISender = class(TELMailSender);
```

Remarks

Implemented by Outlook.

6.4 SMTP

Unit

ESendMailSMTP 129

Syntax

```
Code (Delphi)
TELMailSMTPSender = class(TELMailSMTPBaseSender 166);
```

Remarks

6.4.1 SMTP server

Implements sending via SMTP server protocol.

Unit

ESendMailSMTP 129

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELMailSMTPServerSender = class(TELMailSMTPBaseSender 166);
```

Remarks

Works by creating a fake e-mail server.

6.4.2 SMTP client

Implements sending via SMTP client protocol.

Unit

ESendMailSMTP 129

Syntax

```
Code (Delphi)
TELMailSMTPClientSender = class(TELMailSMTPBaseSender 166);
```

Remarks

Same as normal e-mail client application.

6.5 HTTP(S)

Implements sending via HTTP posting.

Unit

ESendWebHTTP 130

Syntax

```
Code (Delphi)
TELHTTPSender = class(TELWebFilesSender);
```

Remarks

Requires a custom receiver script or form.

6.6 FTP

Implements sending via FTP uploading.

Unit

ESendWebFTP 130

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELFTPSender = class(TELWebFilesSender);
```

Remarks

Application settings must be set up so each bug report has unique file name.

6.7 Web

Mantis 44	Send class for Mantis bug tracker (web-based)
FogBugz 44	Send class for FogBugz bug tracker (web-based)
BugZilla 44	Send class for BugZilla bug tracker (web-based)

See also

- API-based vs. web-based access

6.7.1 Mantis

Enter topic text here.

6.7.2 BugZilla

Enter topic text here.

6.7.3 FogBugz

Enter topic text here.

6.8 API

Mantis 44	Send class for Mantis bug tracker (API-access)
FogBugz 45	Send class for FogBugz bug tracker (API-access)

See also

- API-based vs. web-based access

6.8.1 Mantis

Implements API access to Mantis bug tracker

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ESendAPIMantis 125

Syntax

```
Code (Delphi)
TELTrackerMantisSender = class(TELTrackerSender);
```

Remarks

Mantis uses SOAP API.

Requirements

Bug tracker	Mantis 1.2.7 or above

6.8.2 FogBugz

Implements API access to Fogbugz bug tracker

Unit

ESendAPIFogBugz 122

Syntax

```
Code (Delphi)
TELTrackerFogBugzSender = class(TELTrackerSender);
```

Remarks

Fogbugz uses custom XML API.

Requirements

Bug tracker	Fogbugz 6 or above

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part

7 All Units

EAppCLX 53	This is unit EAppCLX
EAppCrossModule 53	Support unit for EAppDLL and EExceptionManager.
EAppFMX 54	This is unit EAppFMX
EAppHTTPApp 54	This is unit EAppHTTPApp
EAppType 54	General support unit for application type concept (low-level).
EAppVCL 56	This is unit EAppVCL
EAppWebBroker 56	This is unit EAppWebBroker
EBase 57	This is a basic unit for EurekaLog.
EBaseClasses 58	This is unit EBaseClasses
ECallStack 59	Unit for call stack classes and tracing code.
EClasses 61	This is unit EClasses
EClassUtils 61	This is unit EClassUtils
ECompatibility 62	Language forward compatibility support unit.
EComponent 62	This unit contains EurekaLog 7 component.
EConfig 63	Configuration management unit.
EConsts 65	Global constants.
ECore 65	This is unit ECore
EDbgHelp 65	This is unit EDbgHelp
EDebugDWARF 66	This is unit EDebugDWARF
EDebugEL 66	This is unit EDebugEL
EDebugEL7 67	This is unit EDebugEL7

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EDebugExportBase 67	This is unit EDebugExportBase
EDebugExports 67	This is unit EDebugExports
EDebugInfo 68	Debug information unit.
EDebugJCL 69	This is unit EDebugJCL
EDebugMAB 69	This is unit EDebugMAB
EDebugMAB2 69	This is unit EDebugMAB2
EDebugMap 70	This is unit EDebugMap
EDebugME 70	This is unit EDebugME
EDebugMS 70	This is unit EDebugMS
EDebugStabs 71	This is unit EDebugStabs
EDebugTD32 71	This is unit EDebugTD32
EDialog 72	Base unit for dialogs.
EDialogCGI 73	This is unit EDialogCGI
EDialogConsole 73	This is unit EDialogConsole
EDialogHTMLBased 74	This is unit EDialogHTMLBased
EDialogSend 74	This is unit EDialogSend
EDialogSendWinAPI 74	This is unit EDialogSendWinAPI
EDialogService 75	This is unit EDialogService
EDialogWeb 75	This is unit EDialogWeb
EDialogWER 75	This is unit EDialogWER
EDialogWinAPI 76	This is unit EDialogWinAPI
EDialogWinAPIEurekaLog 76	This is unit EDialogWinAPIEurekaLog
EDialogWinAPIEurekaLogDetailed 76	This is unit EDialogWinAPIEurekaLogDetailed

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EDialogWinAPIMSClassic 77	This is unit `EDialogWinAPIMSClassic`
EDialogWinAPIStepsToReproduce 77	This is unit `EDialogWinAPIStepsToReproduce`
EDisAsm 78	Disassembler unit.
EDLLs 78	Support unit for dynamic DLL loading.
EDNS 79	DNS support unit.
EDWARF 79	This is unit `EDWARF`
EEncoding 79	Encoding support unit.
EEncrypt 80	Encryption support unit.
EEvents 83	EurekaLog events.
EException 89	Unit with exception information class.
EExceptionHook 90	This is unit `EExceptionHook`
EExceptionInfo 90	This is unit `EExceptionInfo`
EExceptionInfoBCB 91	This is unit `EExceptionInfoBCB`
EExceptionInfoCLR 91	This is unit `EExceptionInfoCLR`
EExceptionInfoCpp 91	This is unit `EExceptionInfoCpp`
EExceptionInfoDelphi2 92	This is unit `EExceptionInfoDelphi2`
EExceptionInfoDelphiANSI 92	This is unit `EExceptionInfoDelphiANSI`
EExceptionInfoDelphiUnicode 92	This is unit `EExceptionInfoDelphiUnicode`
EExceptionInfoGeneric 92	This is unit `EExceptionInfoGeneric`
EExceptionInfoLLVM 93	This is unit `EExceptionInfoLLVM`
EExceptionInfoMSVC 93	This is unit `EExceptionInfoMSVC`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EExceptionManager 93	No description available
EFileMemory 95	This is unit `EFileMemory`
EFixSafeCallException 95	This is unit `EFixSafeCallException`
EFonts 95	Adaptive fonts support unit.
EFreeze 96	Freeze detection unit.
EHash 97	Hashing support unit.
EHook 99	This is unit `EHook`
EHTMLHelp 99	This is unit `EHTMLHelp`
EInfoFormat 99	This is unit `EInfoFormat`
EInject 100	This is unit `EInject`
EInterfaces 100	This is unit `EInterfaces`
EJSON 100	This is unit `EJSON`
EListView 100	This is unit `EListView`
ELogBuilder 101	Bug report builder unit.
ELogging 103	Logging support unit.
ELogManager 105	This is unit `ELogManager`
ELowLevel 105	Low level routines.
ELowLevelClasses 108	Low-level classes support unit.
ELowLevelExceptions 108	This is unit `ELowLevelExceptions`
EMantisConnectStub 108	This is unit `EMantisConnectStub`
EMAPI 108	This is unit `EMAPI`
EMapParserBase 109	This is unit `EMapParserBase`
EMapWin32 109	This is unit `EMapWin32`
EMemLeaks 109	Unit for diagnosing memory problems (leaks, overflows, etc.).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EMLang 113	MLang headers.
EModules 114	This is unit EModules
EMonitors 114	This is unit EMonitors
ENT 114	This is unit ENT
EOSApiList 115	This is unit EOSApiList
EPChars 115	This is unit EPChars
EPEImage 115	PE(64) formats support unit.
EPNG 115	This is unit EPNG
EResLeaks 116	This is unit EResLeaks
EResourceStrings 116	This is unit EResourceStrings
ERunTime 116	This is unit ERunTime
EScreenShot 119	Screenshot support unit.
ESend 120	Base unit for send methods.
ESendAPIBugZilla 121	This is unit ESendAPIBugZilla
ESendAPIExceptionless 122	This is unit ESendAPIExceptionless
ESendAPIFogBugz 122	This is unit ESendAPIFogBugz
ESendAPIGitHub 123	This is unit ESendAPIGitHub
ESendAPIGitLab 124	This is unit ESendAPIGitLab
ESendAPIJIRA 124	This is unit ESendAPIJIRA
ESendAPIJSON 125	This is unit ESendAPIJSON
ESendAPIMantis 125	This is unit ESendAPIMantis
ESendAPIRedmine 126	This is unit ESendAPIRedmine
ESendAPIYouTrack 126	This is unit ESendAPIYouTrack
ESendMail 127	This is unit ESendMail

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ESendMailMAPI 127	This is unit `ESendMailMAPI`
ESendMailShell 128	This is unit `ESendMailShell`
ESendMailSMAPI 128	This is unit `ESendMailSMAPI`
ESendMailSMTP 129	This is unit `ESendMailSMTP`
ESendWeb 129	This is unit `ESendWeb`
ESendWebFTP 130	This is unit `ESendWebFTP`
ESendWebHTTP 130	This is unit `ESendWebHTTP`
ESignature 131	Unit for working with digital signatures of executable files.
ESOAP 132	This is unit `ESOAP`
ESOAPMantis 132	This is unit `ESOAPMantis`
ESocket 132	This is unit `ESocket`
ESockets 132	This is unit `ESockets`
ESpecificDelphi 133	This is unit `ESpecificDelphi`
ESSLSocket 133	This is unit `ESSLSocket`
EStackTracing 133	Low-level stack tracing code.
EStrConsts 134	String constants.
ESysInfo 134	System information functions.
ETaskDialogs 136	This is unit `ETaskDialogs`
EThreadsManager 136	EurekaLog thread manager unit.
ETools 137	This is unit `ETools`
ETypes 138	Basic EurekaLog types.
EUnmangling 139	This is unit `EUnmangling`
EWCT 139	WCT headers.
EWCTSupport 140	This is unit `EWCTSupport`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EWebTools 140	This is unit EWebTools
EWER 140	WER headers.
EWinCrypt 141	Windows cryptography headers.
EWinSock 141	WinSock headers.
ExceptionLog7 141	Main EurekaLog 7 unit.
EXML 143	This is unit EXML
EXMLBuilder 143	This is unit EXMLBuilder
EXMLRPC 143	This is unit EXMLRPC
EXMLRPCBugZilla 144	This is unit EXMLRPCBugZilla
EZip 144	Compression support unit.
EZLib 145	Compression support unit.
EZLibHeaders 145	This is unit EZLibHeaders

7.1 EAppCLX

This is unit EAppCLX

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.2 EAppCrossModule

Support unit for EAppDLL and EExceptionManager.

Remarks

This unit implements callbacks between DLL and EurekaLog-enabled host.

Warning
Do not use this unit. Use EAppDLL unit in lightweight DLLs and EExceptionManager 93 unit in everything else.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- EAppDLL
- EExceptionManager 93
- EAppType 54
- EInterfaces Unit 100
- ELogging Unit 103
- ETypes Unit 138

7.3 EAppFMX

This is unit EAppFMX

See also

- EAppType Unit 54
- EBase Unit 57
- EConsts Unit 65
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.4 EAppHTTPApp

This is unit EAppHTTPApp

See also

- EAppType Unit 54
- EBase Unit 57
- EConsts Unit 65
- EException Unit 89
- EExceptionHook Unit 90
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.5 EAppType

General support unit for application type concept (low-level).

Remarks

Tip
This is low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

Examples of application's type are: VCL, CLX, VCLX, console, service, CGI, IntraWeb and so on.

An application type is arbitrary non-empty string.

Application may have more than one type in some cases. For example, it can be both console and CGI.

Application type is generic qualifier that affects EurekaLog behaviour. For example, VCL application intercepts

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Application.ShowException and uses window error dialog.

This unit also provides a basic support for error messages, based on application type. For example, console application will write message to console and VCL application will show error message in dialog box.

Warning

If you modify this unit, consider the following guidelines:
- **Do not** add additional unit into uses clause.
- **Do not** add initialization/finalization section.

If you must break the above rules - **make sure that there is no memory allocation during startup**!

See also

- EAppType Routines 55
- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- ETaskDialogs Unit 136
- ETypes Unit 138

7.5.1 Routines

GetApplicationWnd 406	Returns application window (if applicable).
GetMainWnd 424	Returns main window of application (if applicable).
GetModuleType 425	Returns type of loaded executable module, based on analyzing its file header.
GetParentWnd 426	Returns active or top-most window, which can be used as a parent window for modal dialogs.
GetTaskBarWnd 429	Returns window of application, which creates a taskbar button (if applicable).
IsAppType 442	Checks if application is of specified type.
IsBPL 443	Checks if specified module is BPL file.
IsCGI 445	Checks if application is CGI application.
IsCGIConsole 445	Checks if application is CGI console application.
IsCGIWin 446	Checks if application is CGI Windows application.
IsCLX 446	Checks if application is CLX application.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IsConsole 447	Checks if application is console application.
IsCPL 448	Checks if application is control panel application.
IsDLL 449	Checks if specified module is DLL file.
IsEXE 455	Checks if specified module is EXE file.
IsFMX 456	Checks if application is FMX application.
IsGUI 458	Checks if application is GUI application.
IsIntraWeb 459	Checks if application is IntraWeb application.
IsISAPI 459	Checks if application is ISAPI application.
IsModuleLibrary 462	Checks if specified module is DLL or BPL file.
IsService 464	Checks if application is a win32 service application.
IsVCL 469	Checks if application is VCL application.
IsWeb 470	Checks if application is web application.
SetAppType 547	Set a type of this application.

See also

- EAppType Unit 54

7.6 EAppVCL

This is unit EAppVCL

See also

- EAppType Unit 54
- EBase Unit 57
- EConsts Unit 65
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ETypes Unit 138

7.7 EAppWebBroker

This is unit EAppWebBroker

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- EAppHTTPApp Unit 54
- EAppType Unit 54
- EBase Unit 57
- EConsts Unit 65
- EException Unit 89
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.8 EBase

This is a basic unit for EurekaLog.

Remarks

It contains a basic routines to control EurekaLog. Can be included in any application without referencing the entire EurekaLog source code. You can include this unit into your `uses`, if you want to get EurekaLog global state at run-time. Alternative approach is to use conditional defines.

EurekaLog defines the following conditional symbols:

- EUREKALOG
- EUREKALOG7

Examples

Code (Delphi)

```
uses
   EBase;
// ...
if IsEurekaLogInstalled then
begin
   // Do something if EurekaLog is installed
end;
```

Code (Delphi)

```
{$IFDEF EUREKALOG}
    // Do something if EurekaLog is installed
{$ENDIF}
```

See also

- ExceptionLog7 141
- EBase Classes 57
- EBase Routines 58

7.8.1 Classes

TThreadEx 255	Abstract class to create separate threads of execution.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- EBase Unit 57

7.8.2 Routines

BeginThreadEx 325	Spawns a separate thread of execution.
EndThreadEx 386	Terminates the execution of a thread.
ExceptionMessage 397	Retrieves error message for an exception.
HandleException 433	Handles exception.
IsEurekaLogActive 452	Indicates if EurekaLog is active or disabled in this application.
IsExpected 455	Checks if exception is an expected exception.
MarkExpected 480	Marks exception as expected exception.
NameThread 496	Names thread for debugging.
NameThreadForDebugging 496	Alias for `NameThread`, this overrides System.Classes.TThread.NameThreadForDebugging.
RaiseExpected 499	Raises expected exception.
RaiseOuterException 500	Raises an exception and captures the current one.
RaiseOuterExceptionExpected 501	Raises expected exception and captures the current one.
SetEurekaLogState 549	Enables or disables EurekaLog in this application.
SetExceptionMessage 551	Sets or changes error message for an exception.
ShowExpected 557	Shows expected exception.

See also

- EBase Unit 57

7.9 EBaseClasses

This is unit `EBaseClasses`

See also

- EBaseClasses Classes 59

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Types 59
- EAppType Unit 54
- EBase Unit 57
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- EResourceStrings Unit 116
- ETools Unit 137
- ETypes Unit 138

7.9.1 Classes

TELDebugInfoSource 164	Base class for debug information provider.

See also

- EBaseClasses Unit 58

7.9.2 Types

PELLocationInfo 608	A pointer to code location information.
TELLocationInfo 662	A code location information.

See also

- EBaseClasses Unit 58

7.10 ECallStack

Unit for call stack classes and tracing code.

Remarks

Use classes and routines from this unit when you need to build call stack, get caller address, etc.

See also

- EStackTracing unit 133
- EDebugInfo unit 68
- ECallStack Classes 60
- ECallStack Types 61
- ECallStack Routines 60
- ECallStack Globals 60
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EDisAsm Unit 78
- EDlls Unit 78
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- ENT Unit 114
- EResLeaks Unit 116
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETypes Unit 138
- EWCT Unit 139
- EWCTSupport Unit 140

7.10.1 Classes

TEurekaBaseStackList 172	Base call stack class.
TEurekaStackListV6 217	Stack tracing class for EurekaLog 6 method.
TEurekaStackListV7 218	Stack tracing class for EurekaLog 7 RAW method.
TGenericStackList 249	Common stack tracing class.

See also

- ECallStack Unit 59

7.10.2 Globals

DefaultTracerMask 753	Mask for default call stack class.
EurekaCallStackClass 755	Default call stack class.

See also

- ECallStack Unit 59

7.10.3 Routines

GetTracer 430	Get stack tracing class by number.
GetTracerClass 431	Get stack tracing class by number.

See also

- ECallStack Unit 59

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.10.4 Types

TEurekaCallStack 678	Alias for TEurekaStackListV7.
TEurekaCallStackClass 678	Class of TEurekaBaseStackList.
TEurekaStackList 684	Alias for TEurekaStackListV7.

See also

- ECallStack Unit 59

7.11 EClasses

This is unit `EClasses`

See also

- EClasses Classes 61
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EEncrypt Unit 80
- EHash Unit 97
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- EStackTracing Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

7.11.1 Classes

TEurekaModuleOptions 216	EurekaLog options.

See also

- EClasses Unit 61

7.12 EClassUtils

This is unit `EClassUtils`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ELowLevel Unit 105
- ETypes Unit 138

7.13 ECompatibility

Language forward compatibility support unit.

Remarks

This unit contains routines from latest Delphi/C++ Builder RTL.

This unit should be listed as a last unit in `uses` clause of `implementation` section.

See also

- EBaseClasses Unit 58
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.14 EComponent

This unit contains EurekaLog 7 component.

Remarks

EurekaLog component is used as simple way to react to events. This is analog of TApplicationEvents component. It is optional, it is not requited. See description of the component for more info.

See also

- TEurekaLogEvents 33
- Events 11
- EComponent Classes 62
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- EDialog Unit 72
- EEvents Unit 83
- EException Unit 89
- ELogBuilder Unit 101
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESend Unit 120
- ESendWeb Unit 129
- ESysInfo Unit 134
- ETypes Unit 138

7.14.1 Classes

TEurekaLogEvents 33	EurekaLog 7 component.

See also

- EComponent Unit 62

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.15 EConfig

Configuration management unit.

Remarks

This unit contain basic support routines for saving and loading EurekaLog configuration information.

EurekaLog configuration can be stored either in the registry or in .ini-file. This unit encapsulates configuration management, abstracting callers from knowing underlying storage specifics.

EurekaLog code can use ConfigReadXXX and ConfigWriteXXX routines to read and write configuration without worrying about back-end storage. All ConfigReadXXX functions are just wrappers around ConfigReadString 338 function. Similary, all ConfigWriteXXX procedures are just wrappers around ConfigWriteString 342 procedure.

Be default, EurekaLog configuration is stored in the registry under GeneralEurekaLogRegistryKey ('\Software\EurekaLab\EurekaLog\7.0') registry key.

Callers may change this behaviour by calling SetupConfig 552 procedure.

For example, EurekaLog Viewer changes default location to registry key GeneralEurekaLogViewerRegistryKey ('\Software\EurekaLab\Viewer\4.0'). And in standalone mode EurekaLog Viewer changes configuration location to external .ini file.

High-level ConfigXXX functions calls low-level function, depending on current storage's configuration. Low-level routines are RegKeyXXX, IniReadString 435/IniWriteString 440, backward-compatibility routines ReadRawStrings/WriteRawStrings and also a helper functions like EncodeConfigValue 384/DecodeConfigValue 352 and informational routines ConfigIsRegistry 333, ConfigFileName 331, ConfigKeyPath 334.

See also

- EConfig Routines 63
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EEncrypt Unit 80
- EHash Unit 97
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ESysInfo Unit 134
- ETypes Unit 138

7.15.1 Routines

ConfigDelete 331	Deletes a key from the specified section.
ConfigFileName 331	Returns .ini-file name to store EurekaLog configuration.
ConfigIsRegistry 333	Indicates if EurekaLog configuration is stored in the registry or in .ini-file.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ConfigKeyPath 334	Returns registry key name to store EurekaLog configuration.
ConfigReadBool 335	Reads boolean value from specified key and section.
ConfigReadFloat 336	Reads float value from specified key and section.
ConfigReadInteger 337	Reads integer value from specified key and section.
ConfigReadString 338	Reads string value from specified key and section.
ConfigWriteBool 339	Writes boolean value to specified key and section.
ConfigWriteFloat 340	Writes float value to specified key and section.
ConfigWriteInteger 341	Writes integer value to specified key and section.
ConfigWriteString 342	Writes string value to specified key and section.
DecodeConfigValue 352	Low-level routine to decode data after reading from configuration storage.
EncodeConfigValue 384	Low-level routine to encode data before saving to configuration storage.
IniReadString 435	Reads string from .ini file.
IniWriteString 440	Writes string to .ini file.
RegKeyCheck 521	Checks registry key for existing.
RegKeyRead 522	Reads string from registry key.
RegKeyWrite 523	Writes string to registry key.
SetupConfig 552	Changes EurekaLog configuration storage.

See also

- EConfig Unit 63

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.16 EConsts

Global constants.

Remarks

This unit holds global non-localizable constants for EurekaLog.

Warning
DO NOT LOCALIZE STRINGS IN THIS UNIT!

See also

- EStrConsts Unit 134
- EConsts Globals 65
- ETypes Unit 138

7.16.1 Globals

EEurekaLogCurrentVersion 754	Current EurekaLog version (numeric).
EEurekaLogLicense 754	No description available
EEurekaLogVersion 755	Current EurekaLog version (textual).

See also

- EConsts Unit 65

7.17 ECore

This is unit `ECore`

See also

- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ELowLevelExceptions Unit 108
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- EStrConsts Unit 134
- ETypes Unit 138

7.18 EDbgHelp

This is unit `EDbgHelp`

See also

- EBaseClasses Unit 58

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ECompatibility Unit 62
- EDLLs Unit 78
- ELowLevel Unit 105
- EPEImage Unit 115
- ETypes Unit 138

7.19 EDebugDWARF

This is unit EDebugDWARF

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDebugInfo Unit 68
- EDWARF Unit 79
- EEncoding Unit 79
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.20 EDebugEL

This is unit EDebugEL

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugEL7 Unit 67
- EDebugInfo Unit 68
- EEncoding Unit 79
- EEncrypt Unit 80
- EEvents Unit 83
- EFileMemory Unit 95
- EHash Unit 97
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EPEImage Unit 115
- EResLeaks Unit 116
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EZLib Unit 145

7.21 EDebugEL7

This is unit EDebugEL7

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EEncoding Unit 79
- EEncrypt Unit 80
- EEvents Unit 83
- EFileMemory Unit 95
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EZLib Unit 145

7.22 EDebugExportBase

This is unit EDebugExportBase

See also

- EBaseClasses Unit 58
- ECallStack Unit 59
- ECompatibility Unit 62
- EConsts Unit 65
- EDebugInfo Unit 68
- EDisAsm Unit 78
- EFileMemory Unit 95
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EPEImage Unit 115
- EResLeaks Unit 116
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EUnmangling Unit 139

7.23 EDebugExports

This is unit EDebugExports

See also

- ECompatibility Unit 62
- EDebugExportBase Unit 67
- EDebugInfo Unit 68
- ELogging Unit 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 105
- ETypes Unit 138

7.24 EDebugInfo

Debug information unit.

Remarks

This unit contains various routines to work with debug information.

This unit does not implement any support for debug information by itself. You had to also include specific units with support for debug information. For example, EDebugMAP for .map files, EDebugJCL for JEDI (JCL/JDBG), etc. Once support unit is included - you can use this unit (EDebugInfo) to work with debug information. You can include more than one support unit.

See also

- GetLocationInfo function 423
- __FILE__ function
- __FUNCTION__ function
- __LINE__ function
- EDebugInfo Types 68
- EDebugInfo Routines 68
- EAppType Unit 54
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- EThreadsManager Unit 136
- ETypes Unit 138

7.24.1 Routines

GetLocationInfo 423	Returns human-readable information about code location.
GetLocationInfoStr 423	Returns human-readable information about code location.
LocationToStr 478	Converts location information to single line string.

See also

- EDebugInfo Unit 68

7.24.2 Types

TEU... 670	...

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	information should be used for include files.

See also

- EDebugInfo Unit 68

7.25 EDebugJCL

This is unit EDebugJCL

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- ELogging Unit 103
- ELowLevel Unit 105
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.26 EDebugMAB

This is unit EDebugMAB

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EFileMemory Unit 95
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.27 EDebugMAB2

This is unit EDebugMAB2

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ECore Unit 65
- EDebugInfo Unit 68
- EFileMemory Unit 95
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.28 EDebugMap

This is unit EDebugMap

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- ELogging Unit 103
- ELowLevel Unit 105
- EMapParserBase Unit 109
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EUnmangling Unit 139

7.29 EDebugME

This is unit EDebugME

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EEncoding Unit 79
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EZLib Unit 145

7.30 EDebugMS

This is unit EDebugMS

See also

- EAppType Unit 54
- EBase Unit 57

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDbgHelp Unit 65
- EDebugEL Unit 66
- EDebugExportBase Unit 67
- EDebugInfo Unit 68
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETools Unit 137
- ETypes Unit 138

7.31 EDebugStabs

This is unit `EDebugStabs`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EEncoding Unit 79
- ELowLevel Unit 105
- EModules Unit 114
- ENT Unit 114
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EZLib Unit 145

7.32 EDebugTD32

This is unit `EDebugTD32`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- ELogging Unit 103
- ELowLevel Unit 105
- EPEImage Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EUnmangling Unit 139

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.33 EDialog

Base unit for dialogs.

Remarks

This unit contains basic abstract class for implementing error dialogs in EurekaLog. It also registers build-in classes like NULL dialog and MessageBox dialog.

Non-trivial error dialogs are implemented in other units (see below), which refers to this unit (EDialog) and implement one or more child sub-classes.

See also

- EDialogWinAPI 76
- EDialogSendWinAPI 74
- EDialogHTMLBased 74
- EDialogCGI 73
- EDialogConsole 73
- EDialogISAPI
- EDialogService 75
- EDialogWeb 75
- EDialogWER 75
- EDialogSend 74
- EDialog Classes 73
- EDialog Types 73
- EDialog Routines 73
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EEvents Unit 83
- EException Unit 89
- EHash Unit 97
- EInfoFormat Unit 99
- ELogBuilder Unit 101
- ELogging Unit 103
- ELogManager Unit 105
- ELowLevel Unit 105
- ELowLevelExceptions Unit 108
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- EScreenshot Unit 119
- ESend Unit 120
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETaskDialogs Unit 136
- EThreadsManager Unit 136
- ETools Unit 137
- ETypes Unit 138
- EZip Unit 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.33.1 Classes

TBaseDialog 36	Base class for all dialogs.

See also
- EDialog Unit 72

7.33.2 Routines

FindDialogClass 399	Searches dialog class by name.
RegisterDialogClass 502	Registers dialog class.
RegisterDialogClassFirst 503	Registers dialog class.

See also
- EDialog Unit 72

7.33.3 Types

TBaseDialogClass 611	Pointer to dialog class.

See also
- EDialog Unit 72

7.34 EDialogCGI

This is unit `EDialogCGI`

See also
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDialog Unit 72
- EDialogWeb Unit 75
- EException Unit 89
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.35 EDialogConsole

This is unit `EDialogConsole`

See also
- EBaseClasses Unit 58
- ECallStack Unit 59
- ECompatibility Unit 62
- ECore Unit 65

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EDialog Unit 72
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETypes Unit 138

7.36 EDialogHTMLBased

This is unit `EDialogHTMLBased`

See also

- EBaseClasses Unit 58
- ECallStack Unit 59
- ECompatibility Unit 62
- EDebugInfo Unit 68
- EDialog Unit 72
- EEncoding Unit 79
- EException Unit 89
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.37 EDialogSend

This is unit `EDialogSend`

See also

- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ELogging Unit 103
- ELowLevel Unit 105
- ESysInfo Unit 134
- ETypes Unit 138

7.38 EDialogSendWinAPI

This is unit `EDialogSendWinAPI`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDialogSend Unit 74
- EDLLs Unit 78
- EListView Unit 100
- ELogging Unit 103
- ELowLevel Unit 105

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EMonitors Unit 114
- ESysInfo Unit 134
- ETypes Unit 138

7.39 EDialogService

This is unit `EDialogService`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDialog Unit 72
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.40 EDialogWeb

This is unit `EDialogWeb`

See also

- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDebugInfo Unit 68
- EDialog Unit 72
- EDialogHTMLBased Unit 74
- EEncoding Unit 79
- EException Unit 89
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.41 EDialogWER

This is unit `EDialogWER`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDialog Unit 72
- EException Unit 89
- EExceptionHook Unit 90
- EExceptionManager Unit 93
- EInfoFormat Unit 99
- ELogBuilder Unit 101
- ELowLevel Unit 105
- EModules Unit 114
- EPChars Unit 115
- ESysInfo Unit 134
- ETypes Unit 138
- EWER Unit 140

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.42 EDialogWinAPI

This is unit `EDialogWinAPI`

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- ECompatibility Unit 62
- EConsts Unit 65
- EDialog Unit 72
- EDLLs Unit 78
- EFonts Unit 95
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EMonitors Unit 114
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.43 EDialogWinAPIEurekaLog

This is unit `EDialogWinAPIEurekaLog`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDialog Unit 72
- EDialogWinApi Unit 76
- EEvents Unit 83
- EException Unit 89
- EListView Unit 100
- ELogBuilder Unit 101
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EMonitors Unit 114
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.44 EDialogWinAPIEurekaLogDetailed

This is unit `EDialogWinAPIEurekaLogDetailed`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConfig Unit 63

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EConsts Unit 65
- ECore Unit 65
- EDialog Unit 72
- EDialogWinAPI Unit 76
- EDialogWinAPIEurekaLog Unit 76
- EDLLs Unit 78
- EException Unit 89
- EInfoFormat Unit 99
- EListView Unit 100
- ELogBuilder Unit 101
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EMonitors Unit 114
- ENT Unit 114
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.45 EDialogWinAPIMSClassic

This is unit `EDialogWinAPIMSClassic`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDialog Unit 72
- EDialogWinApi Unit 76
- EEvents Unit 83
- EListView Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EMonitors Unit 114
- EResLeaks Unit 116
- ESysInfo Unit 134
- ETypes Unit 138

7.46 EDialogWinAPIStepsToReproduce

This is unit `EDialogWinAPIStepsToReproduce`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDialog Unit 72
- EDialogWinApi Unit 76
- EListView Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EMonitors Unit 114
- EResLeaks Unit 116

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 138

7.47 EDisAsm

Disassembler unit.

Remarks

This unit holds disassembler for current CPU. This unit is used by EurekaLog to disassemble machine code for bug report. Use routines from this unit to disassemble code in the current process.

Note
This unit does not hold any specific implementation. It is just a stub to other units. Implementations for specific CPUs can be found in EDisAsmX8632, EDisAsmX8664, etc.

See also

- EDisAsm Types 78
- EDisAsm Routines 78
- ETypes Unit 138

7.47.1 Routines

DecodeInstruction 353	Decodes a single CPU instruction.
DisasmInstruction 356	Disassebles a single CPU instruction.
GetAsmSize 406	Returns size of a single machine opcode.
GetFunctionSize 422	Returns size of the function in bytes.
InstructionToString 441	Converts decoded CPU instruction to string represenstation.

See also

- EDisAsm Unit 78

7.47.2 Types

PInstruction 608	A pointer to TInstruction 687.
TInstruction 687	A CPU instruction.

See also

- EDisAsm Unit 78

7.48 EDLLs

Support unit for dynamic DLL loading.

Remarks

This unit loads DLL dynamically on request. Just use appropriate Lib*XYZ* function to obtain handle to the DLL.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

DLL will be loaded and unloaded automatically.

See also
- ETypes Unit 138

7.49 EDNS

DNS support unit.

Remarks
This unit contains headers (low-level) and hi-level routines to work with DNS requests.

See also
- EDNS Classes 79
- EDNS Routines 79
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- ELowLevel Unit 105
- EStrConsts Unit 134
- ETypes Unit 138

7.49.1 Classes

EDNSError 151	Base class for DNS errors.

See also
- EDNS Unit 79

7.49.2 Routines

GetMXServerFromEmail 426	Retrieves mail server for e-mail address.

See also
- EDNS Unit 79

7.50 EDWARF

This is unit EDWARF

See also
- ETypes Unit 138

7.51 EEncoding

Encoding support unit.

Remarks
This unit contains routines data encoding.

Note

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Encoding is often confused with encryption and hashing, but they are not the same. All three transform data into another format. Both encoding and encryption are reversible, and hashing is not.

The purpose of encoding is to transform data so that it can be properly (and safely) consumed by a different type of system, e.g. binary data being displayed to human, or being sent over email. The goal is not to keep information secret, but rather to ensure that it's able to be properly consumed.

The purpose of encryption is to transform data in order to keep it secret from others, e.g. sending someone a secret letter that only they should be able to read, or securely sending a password over the Internet. Encryption transforms data into another format in such a way that only specific individual(s) can reverse the transformation. It uses a key, which is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation.

Unlike encoding and encryption - hashing is not reversible process, it will tale arbitrary input and produce fixed-length result, so same input always produce the same output. Additionally, any modification of a given input should result in drastic change to the hash.

Use `EEncoding` unit for encoding, EEncrypt 80 unit for encryption, EHash 97 unit for hasing, ESignature 131 unit for reading certificate info from executables.

See also

- TEAEncrypt Function 572
- RSAEncrypt Function 528
- RSASign Function 536
- MemProtect Function 491
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EMLang Unit 113
- EPChars Unit 115
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.52 EEncrypt

Encryption support unit.

Remarks

This unit contains routines for symmetrical (TEA cipher, Twofish cipher) and asymmetrical (RSA cipher) encryption and decryption, as well as digital signing and verification (RSA cipher).

Note

Encryption is often confused with encoding and hashing, but they are not the same. All three transform data into another format. Both encoding and encryption are reversible, and hashing is not.

The purpose of encryption is to transform data in order to keep it secret from others, e.g. sending someone a secret letter that only they should be able to read, or securely sending a password over the Internet. Encryption transforms data into another format in such a way that only specific individual(s) can reverse the transformation. It uses a key, which is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation.

The purpose of encoding is to transform data so that it can be properly (and safely) consumed by a

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

different type of system, e.g. binary data being displayed to human, or being sent over email. The goal is not to keep information secret, but rather to ensure that it's able to be properly consumed.

Unlike encoding and encryption - hashing is not reversible process, it will tale arbitrary input and produce fixed-length result, so same input always produce the same output. Additionally, any modification of a given input should result in drastic change to the hash.

Use `EEncrypt` unit for encryption, EEncoding 79 unit for encoding, EHash 97 unit for hasing, ESignature 131 unit for reading certificate info from executables.

See also

- TEAEncrypt Function 572
- TwofishEncrypt Function 583
- RSAEncrypt Function 528
- RSASign Function 536
- MemProtect Function 491
- EEncrypt Types 83
- EEncrypt Routines 81
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EEncoding Unit 79
- EHash Unit 97
- ELowLevel Unit 105
- ETypes Unit 138
- EWinCrypt Unit 141

7.52.1 Routines

DataProtect 349	Encrypts data.
DataUnprotect 350	Decrypts data.
EncryptBufferFree 385	Releases the encryption buffer.
InitSalt 438	Generates random salt.
MemProtect 491	Encrypts process memory.
MemUnprotect 493	Decrypts process memory.
RSADecrypt 524	Decrypts encrypted data by using RSA cipher.
RSADestroyKey 527	Deletes RSA key.
RSAEncrypt 528	Encrypts "clear text" data by using RSA cipher.
RSAGenKey 531	Creates new RSA key.
RSALoadPrivateKey 532	Imports private key.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

RSALoadPublicKey 533	Imports public key.
RSASavePrivateKey 534	Exports private key.
RSASavePublicKey 535	Exports public key.
RSASign 536	Signs data.
RSAVerify 538	Verifies digital signature.
SecureFree 545	Wipes and releases memory.
TEADecrypt 559	Decrypts encrypted data by using TEA cipher.
TEADefaultKey 563	Returns pre-initialized default key for TEA.
TEADeriveKey 565	Initializes TEA key from password.
TEADeriveKeyHMAC 569	Initializes TEA key from password via MD5 HMAC.
TEADestroyKey 572	Fills TEA key with zeros.
TEAEncrypt 572	Encrypts "clear text" data by using TEA cipher.
TEAInitSessionKey 576	Generates random TEA key.
TwofishDecrypt 578	Decrypts encrypted data by using Twofish cipher.
TwofishDeriveKey 580	Initializes Twofish key from password.
TwofishDeriveKeyHMAC 582	Initializes Twofish key from password using SHA-256 HMAC.
TwofishDestroyKey 583	Deletes Twofish key.
TwofishEncrypt 583	Encrypts "clear text" data by using Twofish cipher.
TwofishInitIV 585	Creates new IV.
TwofishInitKey 586	"Imports" (precalculates) Twofish key from RAW key value.

See also

- EEncrypt Unit 80

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.52.2 Types

PSalt 609	Pointer to salt.
PTwofishInitVector 610	Pointer to IV for Twofish.
TEncryptBuffer 671	Contains an arbitrary array of bytes.
TRSAExport 717	Defines how to export/import RSA key.
TRSAImport 718	Alias to `TRSAExport`.
TRSAKey 719	Key for RSA encryption.
TRSASignature 720	Digital signature.
TSalt 721	Salt value for deriving key from password.
TTEAKey 737	Key for TEA encryption.
TTwofishInitVector 739	Initialization Vector (IV) for Twofish.
TTwofishKey 740	(Precalculated) Key for Twofish encryption.
TTwofishRAWKey 740	(RAW) Key for Twofish encryption.

See also

- EEncrypt Unit 80

7.53 EEvents

EurekaLog events.

Remarks

This unit serves event for EurekaLog. It contains event types, allows you to register (subscribe to) events, and raise events manually (if ever needed).

Each event is done as either method or procedure. You can use either option, there is no difference between method and procedure, so you can use whatever you want.

All events follows the same template. To use EurekaLog event you must declare a procedure or method with required signature (see explanation of specific events for exact code syntax). There is RegisterEvent*EventName* procedure, which you can use to register event handler for any event. And there is reverse UnregisterEvent*EventName* routine, which cancels event for your handler.

Note
Additionaly, there is also DoEvent*EventName* procedure, which can be used to invoke event, but it's typically used by EurekaLog itself.

See examples below for sample code to register event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Instead of using events from code - you can subscribe to event by using EurekaLog component 33. You can just drop component to form and double click on any event to create event handler - much like you do with creating event handlers for, say, FormCreate event for form, etc.

Tip
It's recommended to use registration from code instead of using the component.

For detailed description of each event - please see this index article 11.

Caution!
Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Examples

The common example of using an event hander (in this example OnExceptionNotify 14 event is used) is as following.

Example for function handler:

Code (Delphi)

```
unit Unit1;
...
implementation
...
procedure MyHandler(const ACustom: Pointer; AExceptionInfo: TEurekaExcep
begin
   // ... your code ...
end;
initialization
   RegisterEventExceptionNotify(nil, MyHandler);
finalization
   UnregisterEventExceptionNotify(nil, MyHandler); // optional, may be m
end.
```

Code (Delphi)

```
unit Unit1;
...
type
   TForm1 = class
   ...
      procedure MyHandler(AExceptionInfo: TEurekaExceptionInfo; var AHar
   end;
implementation
...
procedure TForm1.FormCreate(Sender: TObject);
begin
   RegisterEventExceptionNotify(MyHandler);
   ...
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
   UnregisterEventExceptionNotify(MyHandler); // mandatory, must be pres
   ...
end;
procedure TForm1.MyHandler(AExceptionInfo: TEurekaExceptionInfo; var AHa
begin
   // ... your code ...
end;
...
end.
```

See also

- EurekaLog component 33
- Events 11
- EEvents Types 87
- EEvents Routines 85
- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EException Unit 89
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ETypes Unit 138

7.53.1 Routines

AddCustomDataToBugReport 319	Adds static data to bug report.
AddCustomWebField 320	Adds static data to web requests.
RegisterEventAttachedFilesRequest 504	Registers OnAttachedFilesRequest event handler.
RegisterEventBeginReportGen 505	Registers OnBeginReportGen event handler.
RegisterEventCustomBugID 506	Registers OnCustomBugID event handler.
RegisterEventCustomButtonClick 507	Registers OnCustomButtonClick event handler.
RegisterEventCustomDataRequest 508	Registers OnCustomDataRequest event handler.
RegisterEventCustomFileName 509	Registers OnCustomFileName event handler.
RegisterEventCustomWebFieldsRequest 510	Registers OnCustomWebFieldRequest event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

RegisterEventEndReportGen 511	Registers OnEndReportGen event handler.
RegisterEventExceptionAction 512	Registers OnExceptionAction event handler.
RegisterEventExceptionError 513	Registers OnExceptionError event handler.
RegisterEventExceptionNotify 514	Registers OnExceptionNotify event handler.
RegisterEventHelpButtonClick 515	Registers OnHelpButtonClick event handler.
RegisterEventPasswordRequest 516	Registers OnPasswordRequest event handler.
RegisterEventRaise 517	Registers OnRaise event handler.
RegisterEventReproduceRequest 518	Registers OnReproduceRequest event handler.
RegisterEventZippedFilesRequest 519	Registers OnZippedFilesRequest event handler.
UnRegisterEventAttachedFilesRequest 587	Unregisters previously registered OnAttachedFilesRequest event handler.
UnRegisterEventBeginReportGen 588	Unregisters previously registered OnBeginReportGen event handler.
UnRegisterEventCustomBugID 589	Unregisters previously registered OnCustomBugID event handler.
UnRegisterEventCustomButtonClick 589	Unregisters previously registered OnCustomButtonClick event handler.
UnRegisterEventCustomDataRequest 590	Unregisters previously registered OnCustomDataRequest event handler.
UnRegisterEventCustomFileName 591	Unregisters previously registered OnCustomFileName event handler.
UnRegisterEventCustomWebFieldsRequest 592	Unregisters previously registered OnCustomWebFieldRequest event handler.
UnRegisterEventEndReportGen 593	Unregisters previously registered OnEndReportGen event handler.
UnRegisterEventExceptionAction 594	Unregisters previously registered

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	OnExceptionAction event handler.
UnRegisterEventExceptionError 594	Unregisters previously registered OnExceptionError event handler.
UnRegisterEventExceptionNotify 595	Unregisters previously registered OnExceptionNotify event handler.
UnRegisterEventHelpButtonClick 596	Unregisters previously registered OnHelpButtonClick event handler.
UnRegisterEventPasswordRequest 597	Unregisters previously registered OnPasswordRequest event handler.
UnRegisterEventRaise 598	Unregisters previously registered OnRaise event handler.
UnRegisterEventReproduceRequest 598	Unregisters previously registered OnReproduceRequest event handler.
UnRegisterEventZippedFilesRequest 599	Unregisters previously registered OnZippedFilesRequest event handler.

See also

- EEvents Unit 83

7.53.2 Types

TELEvAttachedFilesRequestMeth 21	Method event handler for OnAttachedFilesRequest event.
TELEvAttachedFilesRequestProc 625	Procedural event handler for OnAttachedFilesRequest event.
TELEvCustomBugIDMeth 25	Method event handler for OnCustomBugID event.
TELEvCustomBugIDProc 626	Procedural event handler for OnCustomBugID event.
TELEvCustomButtonClickMeth 28	Method event handler for OnCustomButtonClick event.
TELEvCustomButtonClickProc 628	Procedural event handler for OnCustomButtonClick event.
TELEvCustomDataRequestMeth 26	Method event handler for OnCustomDataRequest event.
TELEvCustomDataRequestProc 631	Procedural event handler for

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	OnCustomDataRequest event.
TELEvCustomFileNameMeth 24	Method event handler for OnCustomFileName event.
TELEvCustomFileNameProc 633	Procedural event handler for OnCustomFileName event.
TELEvCustomWebFieldsRequestMeth 27	Method event handler for OnCustomWebFieldsRequest event.
TELEvCustomWebFieldsRequestProc 635	Procedural event handler for OnCustomWebFieldsRequest event.
TELEvExceptionActionMeth 16	Method event handler for OnExceptionAction event.
TELEvExceptionActionProc 638	Procedural event handler for OnExceptionAction event.
TELEvExceptionErrorMeth 18	Method event handler for OnExceptionError event.
TELEvExceptionErrorProc 642	Procedural event handler for OnExceptionError event.
TELEvExceptionNotifyMeth 14	Method event handler for OnExceptionNotify event.
TELEvExceptionNotifyProc 645	Procedural event handler for OnExceptionNotify event.
TELEvHelpButtonClickMeth 29	Method event handler for OnHelpButtonClick event.
TELEvHelpButtonClickProc 648	Procedural event handler for OnHelpButtonClick event.
TELEvPasswordRequestMeth 19	Method event handler for OnPasswordRequest event.
TELEvPasswordRequestProc 651	Procedural event handler for OnPasswordRequest event.
TELEvRaiseMeth 12	Method event handler for OnRaise event.
TELEvRaiseProc 653	Procedural event handler for OnRaise event.
TELEvReportGenMeth 30	Method event handler for OnReportBeginGen/ OnReportEndGen events.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TELEvReportGenProc 657	Procedural event handler for OnReportBeginGen/ OnReportEndGen event.
TELEvReproduceRequestMeth 23	Method event handler for OnReproduceRequest event.
TELEvReproduceRequestProc 659	Procedural event handler for OnReproduceRequest event.
TELEvZippedFilesRequestMeth 22	Method event handler for OnZippedFilesRequest event.
TELEvZippedFilesRequestProc 661	Procedural event handler for OnZippedFilesRequest event.

See also

- EEvents Unit 83

7.54 EException

Unit with exception information class.

Examples

This unit holds TEurekaExceptionInfo 180 class. This is class for storing exception information. It's used everywhere in EurekaLog when you need to work with exceptions.

See also

- TEurekaExceptionInfo class 180
- ExceptionManager function 396
- EException Classes 90
- EAppType Unit 54
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EEvents Unit 83
- EExceptionInfo Unit 90
- EHash Unit 97
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.54.1 Classes

TEurekaExceptionInfo 180	Exception information class.

See also

- EException Unit 89

7.55 EExceptionHook

This is unit `EExceptionHook`

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EDLLs Unit 78
- EException Unit 89
- EExceptionInfo Unit 90
- EExceptionManager Unit 93
- EHook Unit 99
- EInject Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- EThreadsManager Unit 136
- ETypes Unit 138

7.56 EExceptionInfo

This is unit `EExceptionInfo`

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfoBCB Unit 91
- EExceptionInfoCLR Unit 91
- EExceptionInfoCpp Unit 91
- EExceptionInfoDelphi2 Unit 92
- EExceptionInfoDelphiANSI Unit 92
- EExceptionInfoDelphiUnicode Unit 92
- EExceptionInfoGeneric Unit 92
- EExceptionInfoLLVM Unit 93
- EExceptionInfoMSVC Unit 93
- ELogging Unit 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.57 EExceptionInfoBCB

This is unit `EExceptionInfoBCB`

See also

- EAppType Unit 54
- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.58 EExceptionInfoCLR

This is unit `EExceptionInfoCLR`

See also

- EAppType Unit 54
- EBase Unit 57
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.59 EExceptionInfoCpp

This is unit `EExceptionInfoCpp`

See also

- EAppType Unit 54
- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.60 EExceptionInfoDelphi2

This is unit `EExceptionInfoDelphi2`

See also

- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.61 EExceptionInfoDelphiANSI

This is unit `EExceptionInfoDelphiANSI`

See also

- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.62 EExceptionInfoDelphiUnicode

This is unit `EExceptionInfoDelphiUnicode`

See also

- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.63 EExceptionInfoGeneric

This is unit `EExceptionInfoGeneric`

See also

- ECompatibility Unit 62
- EConsts Unit 65

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- EStrConsts Unit 134
- ETypes Unit 138

7.64 EExceptionInfoLLVM

This is unit `EExceptionInfoLLVM`

See also

- EAppType Unit 54
- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138
- EUnmangling Unit 139

7.65 EExceptionInfoMSVC

This is unit `EExceptionInfoMSVC`

See also

- EAppType Unit 54
- EBase Unit 57
- EClasses Unit 61
- ECompatibility Unit 62
- EDLLs Unit 78
- EExceptionInfo Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ETypes Unit 138
- EUnmangling Unit 139

7.66 EExceptionManager

See also

- ExceptionManager function 396
- TExceptionManager.Instance class function 233
- EExceptionManager Classes 94
- EExceptionManager Types 94
- EExceptionManager Routines 94
- EAppCrossModule Unit 53
- EAppType Unit 54
- EBase Unit 57

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugEL Unit 66
- EDebugInfo Unit 68
- EDialog Unit 72
- EEvents Unit 83
- EException Unit 89
- EExceptionHook Unit 90
- EExceptionInfo Unit 90
- EFreeze Unit 96
- EHook Unit 99
- EInject Unit 100
- EInterfaces Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- EStackTracing Unit 133
- EThreadsManager Unit 136
- ETypes Unit 138
- EWCT Unit 139

7.66.1 Classes

TExceptionManager 232	Exception manager class.

See also

- EExceptionManager Unit 93

7.66.2 Routines

ExceptionManager 396	Exception manager.

See also

- EExceptionManager Unit 93

7.66.3 Types

TExceptionNotifyMode 684	Specifies work mode of exception manager.

See also

- EExceptionManager Unit 93

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.67 EFileMemory

This is unit `EFileMemory`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETools Unit 137
- ETypes Unit 138

7.68 EFixSafeCallException

This is unit `EFixSafeCallException`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EHook Unit 99
- ELowLevel Unit 105
- ETypes Unit 138

7.69 EFonts

Adaptive fonts support unit.

Remarks

This unit provides information about system fonts. You can use this unit to use user's customized fonts.

Note
Delphi applications use fixed build-in font names by default. Such applications are not affected by system font settings.

See also

- EFonts Routines 95
- EBaseClasses Unit 58
- ETypes Unit 138

7.69.1 Routines

GUIFont 432	Returns font for GUI applications.
MonoFont 495	Returns fixed-width font.

See also

- EFonts Unit 95

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.70 EFreeze

Freeze detection unit.

Remarks

This unit holds code for running freeze detection checks.

Freeze detection is initialized automatically on startup by invoking InitCheckFreeze 436 function. This function uses FreezeThreadClass 756 variable to select specific hang detection code.

See also

- InitCheckFreeze Function 436
- FreezeThreadClass Variable 756
- EFreeze Classes 96
- EFreeze Types 97
- EFreeze Routines 97
- EFreeze Globals 96
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EDLLs Unit 78
- EException Unit 89
- EExceptionInfo Unit 90
- EExceptionManager Unit 93
- ELogging Unit 103
- ELowLevel Unit 105
- ENT Unit 114
- EStackTracing Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETypes Unit 138
- EWCT Unit 139
- EWCTSupport Unit 140

7.70.1 Classes

TFreezeThread 246	Base class for freeze detection threads.
TMainThreadFreezeDetectionThread 254	Thread class to detects hangs in UI thread.
TWCTFreezeDetectionThread 265	Thread class to detects deadlocks between threads.

See also

- EFreeze Unit 96

7.70.2 Globals

FreezeThreadClass 756	Determinate class to be used as

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	background freeze detection thread.

See also

- EFreeze Unit 96

7.70.3 Routines

DoneCheckFreeze 357	Stops background freeze detection thread.
FreezeNotifyStillAlive 401	Lets EurekaLog know that your main thread is still working.
InitCheckFreeze 436	Starts checking for freezes and hangs.
IsFreezeCheckEnabled 457	Indicate if freeze detection is currently active.
IsFreezeCheckInstalled 458	Indicate if freeze detection was initialized.
PauseFreezeCheck 497	Temporarily disables freeze checks.
RaiseFreezeException 499	Raises freeze/hang/deadlock exception for the specified thread.
ResumeFreezeCheck 524	Enables freeze checks back.

See also

- EFreeze Unit 96

7.70.4 Types

TFreezeThreadClass 686	Class of freeze detection thread.

See also

- EFreeze Unit 96

7.71 EHash

Hashing support unit.

Remarks

This unit contains functions to calculate CRC32, CRC16, MD5, SHA-1, and SDBM hashes.

Note
Hashing is often confused with encryption and encoding, but they are not the same. All three transform data into another format. Both encoding and encryption are reversible, and hashing is not.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unlike encoding and encryption - hashing is not reversible process, it will tale arbitrary input and produce fixed-length result, so same input always produce the same output. Additionally, any modification of a given input should result in drastic change to the hash.

The purpose of encoding is to transform data so that it can be properly (and safely) consumed by a different type of system, e.g. binary data being displayed to human, or being sent over email. The goal is not to keep information secret, but rather to ensure that it's able to be properly consumed.

The purpose of encryption is to transform data in order to keep it secret from others, e.g. sending someone a secret letter that only they should be able to read, or securely sending a password over the Internet. Encryption transforms data into another format in such a way that only specific individual(s) can reverse the transformation. It uses a key, which is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation.

Use EHash unit for hasing, EEncrypt 80 unit for encryption, EEncoding 79 unit for encoding, ESignature 131 unit for reading certificate info from executables.

See also

- EHash Types 345
- EHash Routines 98
- EBaseClasses Unit 58
- ECompatibility Unit 62
- ELowLevel Unit 105
- ETypes Unit 138

7.71.1 Routines

CRC16Hash 345	Calculates CRC-16 checksum.
CRC32Hash 346	Calculates CRC-32 checksum.
MD5Hash 480	Calculates MD5 hash.
MD5HMAC 481	Calculates HMAC-MD5.
SDBMHash 545	Calculates SDBM hash.
SecureFree 547	No description available
SHA1Hash 553	Calculates SHA-1 hash.
SHA1HMAC 554	Calculates HMAC-SHA1.
SHA256Hash 555	Calculates SHA-2/SHA-256 hash.
SHA256HMAC 556	Calculates HMAC-SHA256.

See also

- EHash Unit 97

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.71.2 Types

TCRC16Hash 621	CRC-16 checksum.
TCRC32Hash 622	CRC-32 checksum.
TMD5Hash 705	MD5 hash.
TSDBMHash 722	SDBM hash.
TSHA1Hash 730	SHA-1 hash.
TSHA256Hash 730	SHA-2/SHA-256 hash.

See also

- EHash Unit 97

7.72 EHook

This is unit `EHook`

See also

- EAppType Unit 54
- EClassUtils Unit 61
- EConsts Unit 65
- EDisAsm Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- ELowLevelClasses Unit 108
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.73 EHTMLHelp

This is unit `EHTMLHelp`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- ETypes Unit 138

7.74 EInfoFormat

This is unit `EInfoFormat`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ELogging Unit 103
- ELowLevel Unit 105
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.75 EInject

This is unit `EInject`

See also

- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EHook Unit 99
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ENT Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ESysInfo Unit 134
- ETypes Unit 138

7.76 EInterfaces

This is unit `EInterfaces`

See also

- ETypes Unit 138

7.77 EJSON

This is unit `EJSON`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EInfoFormat Unit 99
- ELowLevel Unit 105
- ETypes Unit 138

7.78 EListView

This is unit `EListView`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- ELowLevel Unit 105

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 138

7.79 ELogBuilder

Bug report builder unit.

Remarks

This unit holds declaration for abstract log builder class (TBaseLogBuilder 152) and 3 build-in specific implementation of log builder class (TLogBuilder 253, TXMLLogBuilder 265, TJSONLogBuilder 252, and TDummyLogBuilder 159).

Log builder class is used by EurekaLog to create bug reports from exception information. Default class to use is defined by LogBuilderClass 757 global variable.

Note
Bug report is a simple multi-line text. Multi-files bug reports (ELP/ZIP) are created by dialog class 36.

See also

- ELogBuilder Classes 101
- ELogBuilder Types 102
- ELogBuilder Routines 102
- ELogBuilder Globals 102
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDebugInfo Unit 68
- EDisAsm Unit 78
- EDLLs Unit 78
- EEncoding Unit 79
- EEvents Unit 83
- EException Unit 89
- EExceptionInfo Unit 90
- EHash Unit 97
- EHook Unit 99
- EInfoFormat Unit 99
- EJSON Unit 100
- ELogging Unit 103
- ELogManager Unit 105
- ELowLevel Unit 105
- EModules Unit 114
- ESpecificDelphi Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETools Unit 137
- ETypes Unit 138
- EXMLBuilder Unit 143

7.79.1 Classes

TBaseLogBuilder 152	Base class for log builder.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TDummyLogBuilder 159	Dummy log builder class.
TJSONLogBuilder 252	JSON EurekaLog 7+ log builder.
TLogBuilder 253	Default EurekaLog 7 log builder.
TXMLLogBuilder 265	XML EurekaLog 7+ log builder.

See also

- ELogBuilder Unit 101

7.79.2 Globals

LogBuilderClass 757	Specifies global log builder class to use.

See also

- ELogBuilder Unit 101

7.79.3 Routines

BuildBugReport 328	Builds bug report from exception information.
DisasmFunction 355	Disasms function.
FindLowAddress 400	Tool function to obtain starting address to disassemble code block around known address.

See also

- ELogBuilder Unit 101

7.79.4 Types

TELEvReportGenMeth 655	Method event handler for OnReportBeginGen/OnReportEndGen events.
TELEvReportGenProc 657	Procedural event handler for OnReportBeginGen/OnReportEndGen event.

See also

- ELogBuilder Unit 101

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.80 ELogging

Logging support unit.

Remarks

This unit contains various routines for logging.

First, you have to create new log file by calling / ELogOpen 378 function.

Next, you add log entries by calling various / ELog 360 -like functions.

Final log will be automatically attached to packed bug report (.elp).

See also

- ELogOpen Function 378
- ELog Function 360
- ELogEnter Function 360
- ELogging Interfaces 103
- ELogging Types 105
- ELogging Routines 103
- EConsts Unit 65
- EDLLs Unit 78
- ELowLevel Unit 105
- EPChars Unit 115
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.80.1 Interfaces

IEurekaLogger 278	Interface which holds a single entry in log file.
IEurekaLoggerParams 295	Sub-routine log entry.

See also

- ELogging Unit 103

7.80.2 Routines

ELog 360	Creates a new log entry.
ELogAddCheckPoint 364	Logs a checkpoint.
ELogAddResetSeparator 364	Resets identation and adds a separator to the log.
ELogAddSeparator 365	Adds a separator to the log.
ELogAssigned 365	Checks if object is assigned and logs result.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ELogAssignedPtr 366	Checks if pointer is assigned and logs result.
ELogClear 367	Clears log file.
ELogCurrency 367	ELog for `Currency` type.
ELogDateTime 368	ELog for `TDateTime` type.
ELogEnter 369	Creates a new log entry about entering method/function/procedure.
ELogError 371	Adds an error to log file.
ELogException 371	ELog for `Exception` type.
ELogFlush 372	Flushes log to disk.
ELogIf 373	Adds a new log entry only if condition is true.
ELogIfDateTime 374	ELogIf for `TDateTime` type.
ELogInterval 375	Add a new time measurement to log file.
ELogMemoryAsHex 376	Adds a memory dump to log.
ELogNote 377	Adds a note to log file.
ELogOpen 378	Creates new log file.
ELogPointer 379	ELog for `Pointer` type.
ELogReminder 380	Adds a reminder to log file.
ELogResetCheckPoint 380	Resets checkpoint counter.
ELogWarning 381	Adds a warning to log file.
ELogWindowHandle 382	ELog for window handle.
ELogWinError 383	ELog for system error codes.
GetPerformanceCount 427	Returns timestamp.

See also

- ELogging Unit 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.80.3 Types

TLogMsgContent 691	Kind of data (content).
TLogMsgType 691	Kind of log entry.
TLogType 692	Common "type".

See also

- ELogging Unit 103

7.81 ELogManager

This is unit `ELogManager`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EEncrypt Unit 80
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ESysInfo Unit 134
- ETypes Unit 138

7.82 ELowLevel

Low level routines.

Remarks

This units consists of low-level routines.

This unit holds very simple general purpose routines. It's main purpose to be basic service unit. It can not have complex code nor complex unit dependency, so it can be used everywhere - including leaks-detection units.

Since it's a very low-level unit - you rarely need to call its routines.

However, this unit also holds few compatibility declarations - i.e. the procedures and functions that are missing in old languages. They are declared here, so code may use new routines even in (for example) Delphi 4.

Warning
If you modify this unit, consider the following guidelines: • **Do not** add additional unit into uses clause. • **Do not** add initialization/finalization section. If you must break the above rules - **make sure that there is no memory allocation during startup**!

See also

- ELowLevel Routines 106

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ENT Unit 114
- EPChars Unit 115
- ESpecificDelphi Unit 133
- ETypes Unit 138
- EWER Unit 140

7.82.1 Routines

AllocatePageAlign 321	No description available
AllocationGranularity 322	No description available
CanCallMemWipeStack 328	Indicates if the `MemWipeStack` function should be called.
CheckELMagicCode 329	No description available
ConvertAddress 343	Convert address to code to real address of code.
ConvertRetAddr 344	Convert return address to instruction pointer.
CurrentInstance 348	Returns current instance.
DebugBreak 352	Causes a breakpoint exception to occur in the current process.
EnsureThreadStack 387	Sets the minimum size of the stack associated with the calling thread or fiber that will be available during any stack overflow exceptions. This is useful for handling stack overflow exceptions; the application can safely use the specified number of bytes during exception handling.
GetReadableSize 428	Verifies that the caller has specified access to the specified range of memory.
IfElse 435	A replacement for ternary `if` operator.
IntoIDE 442	No description available
IsCBuilder 444	Indicates if current project was compiled by C++ Builder.
IsCompiledWithPackages 447	Checks if this module is compiled with packages.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IsDebugged 449	Determines whether the calling process is being debugged by a user-mode debugger.
IsDynamicRTL 450	Indicates if current project was compiled with dynamic RTL DLL.
IsLLVM 460	Indicates if current project is compiled by "new" C++ Builder compiler (e.g. LLVM).
IsReadableFile 463	Verifies that the caller has read access to the specified file.
IsRTLCompiledWithStackFrames 463	Determines whether the RTL was compiled with Stack Frames option enabled.
IsStackFramesProc 464	Determines whether the specified function has a stack frame.
IsValidBlockAddr 465	Verifies that the caller has read access to the specified range of memory.
IsValidClass 466	Verifies that the specified pointer is a valid class.
IsValidCodeAddr 467	Verifies that the caller has execute access to the specified range of memory.
IsValidHandle 468	Verifies that this is a valid kernel handle.
IsValidObject 468	Verifies that the specified pointer is a valid object.
IsVirtualized 470	Checks if application was instrumented/ virtualized.
IsWritableFile 477	Verifies that the caller has write access to the specified file.
MainThreadTopOfStack 479	Returns top of the stack for the main thread.
MemWipeStack 494	Clears unused stack of the current thread.
PageSize 497	No description available
PrimaryInstance 498	The same as MainInstance. See remarks for more info.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

SafeExec 540	Executes routine with exception handling. Any exception is handled as fatal exception.

See also

- ELowLevel Unit 105

7.83 ELowLevelClasses

Low-level classes support unit.

Remarks

This unit contains reduced implementation of some classes from Classes unit. This unit does not have reference to SysUtils unit - so it can be used in other low-level units (such as memory leaks checking).

7.84 ELowLevelExceptions

This is unit `ELowLevelExceptions`

See also

- EConsts Unit 65
- ELowLevel Unit 105
- ETypes Unit 138

7.85 EMantisConnectStub

This is unit `EMantisConnectStub`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EInfoFormat Unit 99
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ESOAPMantis Unit 132
- ETools Unit 137
- ETypes Unit 138
- EXML Unit 143
- EXMLRPC Unit 143

7.86 EMAPI

This is unit `EMAPI`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ESysInfo Unit 134
- ETypes Unit 138

7.87 EMapParserBase

This is unit `EMapParserBase`

See also

- EMapParserBase Classes 109
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EFileMemory Unit 95
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138

7.87.1 Classes

TELAbstractMapParser 159	Base class for .map file parsers.

See also

- EMapParserBase Unit 109

7.88 EMapWin32

This is unit `EMapWin32`

See also

- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EHook Unit 99
- ELowLevel Unit 105
- ETypes Unit 138

7.89 EMemLeaks

Unit for diagnosing memory problems (leaks, overflows, etc.).

Remarks

This unit contains code for EurekaLog's memory manager filter. This filter allows EurekaLog to check for memory problems, such as leaks, overflows, double-free, cross-module free, etc.

This unit does not contains implementation of real memory manager. This is memory filter only. It uses already installed memory manager to allocate and dispose memory.

EurekaLog's memory filter will add a little overhead for each memory allocation. EurekaLog stores additional header with call stack and other service information with each memory block. This header allows EurekaLog to check for memory problems.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Disabling memory leaks checks will significantly decrease size of added overhead (by removing call stack, name, etc.), but not remove overhead completely.

Additionally, a memory filter from EMemLeaks unit provide a way to associate exception object with exception information 180 without using hooks. This allows EurekaLog to implement a clean way to exception tracing - which is useful for protection software. That is why it is very important to try to keep "Enable extended memory manager" option checked - even if you disable all other memory options.

Most routines from EMemLeaks unit should not be called from your code. Exceptions to this rule are:

- IsMemLeaksInstalled 461/IsMemLeaksEnabled 461 functions;
- EnableMemLeaksRegistering 383/DisableMemLeaksRegistering 354 functions;
- EurekaRegisterExpectedMemoryLeak 392/EurekaUnregisterExpectedMemoryLeak 395 functions;
- AllocMemSizeEx 324/AllocMemCountEx 322 functions;
- AllocMemSizeMax 324/AllocMemCountMax 323 functions;
- MemLeaksName function 483;
- MemLeaksOwn function 485;
- MemLeaksSetParentBlock function 489;
- CheckHeap function 330;
- DumpAllocationsToFile function 358;
- EurekaTryGetMem function 394;

Note

EMemLeaks unit is always included in uses clause when EurekaLog is enabled. Even if memory leaks and problems checks are disabled.

That is because memory checks can be enabled at run-time (programmatically or via command-line switches).

Note

EMemLeaks unit is included in uses clause when "Enable extended memory manager" option is checked.

Tip

"Enable extended memory manager" option can be enabled without enabling EurekaLog (via "Activate EurekaLog" option). This allows you to have EurekaLog memory manager for DLLs compiled without EurekaLog (for example, when you use "DLL" profile).

Warning

EMemLeaks unit must be a first unit to be initialized in your application.

There must be no active memory allocations before its initialization.

For this reason this unit must be a first unit in uses clause of your project.

Important!

It is highly important to have compatible memory managers in all modules which use shared memory manager. Therefore, if you are going to use shared memory manager - it is best to enable EMemLeaks unit in all your DLLs and EXE.

Pay close attention to statically linked DLLs, as such DLLs are loaded and initialized **before** main EXE. This can lead to initialization of some units before EMemLeaks unit.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Important!

You can place the following units before EMemLeaks unit in uses clause of your project:
- units with memory manager's implementation (such as FastMM4, SafeMM, etc.);
- units without initialization section;
- units with initialization section which does not allocate memory;
- units with initialization section which does allocate memory, but releases all allocated memory before return.

Normally EurekaLog would not allow any non-EurekaLog unit to be placed before EMemLeaks unit. However, EurekaLog has list of exceptions for this rule. For example, EurekaLog would allow you to place FastMM, FastMM3, FastMM4, SafeMMInstall, ShareMem, FastShareMem, SimpleShareMem, FastMove, FastCode, FastObj, FastSys, etc. units before EMemLeaks unit. You can instruct EurekaLog to allow other units to be placed before EMemLeaks unit. To do that:
1. Go to "Project" / "EurekaLog options" IDE menu;
2. Go to "Advanced" / "Custom / Manual" section;
3. Add "PreInitUnits" option (without quotes) to any location;
4. Add your units to the value of this option. Separate unit names with comma (","). Do not use spaces.

See also

- EMemLeaks Types 113
- EMemLeaks Routines 111
- EConsts Unit 65
- EDisAsm Unit 78
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- ELowLevelExceptions Unit 108
- EPChars Unit 115
- EStackTracing Unit 133
- ETypes Unit 138

7.89.1 Routines

AllocMemCountEx 322	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemCountMax 323	Represents the maximum number of allocated memory blocks.
AllocMemSizeEx 324	Represents the total size of allocated memory blocks.
AllocMemSizeMax 324	Represents the maximum size of allocated memory.
CheckHeap 330	Checks heap for consistency.
DisableMemLeaksRegistering 354	Disables memory leaks checks.
DumpAllocationsToFile 358	Logs all allocated memory to file.
EnableMemLeaksRegistering 383	Enables memory leaks checks.
EurekaAllocMem 388	AllocMem allocates a memory block and

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	initializes each byte to zero.
EurekaFreeMem 389	Frees a memory block previously allocated with EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.
EurekaGetMem 390	Allocates a memory block via EurekaLog memory filter.
EurekaReallocMem 391	Resizes a memory block.
EurekaRegisterExpectedMemoryLeak 392	Register an expected memory leak.
EurekaRegisterExpectedMemoryLeakDynArray 393	Register an expected memory leak (variant for dynamic arrays).
EurekaRegisterExpectedMemoryLeakStr 393	Register an expected memory leak (variant for strings).
EurekaTryGetMem 394	Allocates a memory block via EurekaLog memory filter (w/o error checking).
EurekaUnregisterExpectedMemoryLeak 395	Removes memory block from excluded leaks.
EurekaUnregisterExpectedMemoryLeakDynArray 395	Removes memory block from excluded leaks (variant for dynamic arrays).
EurekaUnregisterExpectedMemoryLeakStr 396	Removes memory block from excluded leaks (variant for strings).
IsMemLeaksEnabled 461	Indicate if memory leaks checks are enabled.
IsMemLeaksInstalled 461	No description available
MemLeaksGetName 482	No description available
MemLeaksName 483	Names a memory block.
MemLeaksNameDynArray 484	Names a memory block (variant for dynamic arrays).
MemLeaksNameStr 485	Names a memory block (variant for strings).
MemLeaksOwn 485	Sets ownership for memory block.
MemLeaksOwnDynArrayBoth 486	Sets ownership for memory block (variant for dynamic arrays).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

MemLeaksOwnDynArrayChild 487	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayParent 487	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnStr 488	Sets ownership for memory block (variant for strings).
MemLeaksSetParentBlock 489	No description available
MemLeaksSetParentBlockDynArrayBoth 489	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayChild 489	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayParent 490	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockStr 491	Sets parent-child relation between two memory blocks for grouping (variant for strings).

See also

- EMemLeaks Unit 109

7.89.2 Types

TDumpCompareFunc 622	Comparator for memory blocks.
TDumpFilterFunc 623	Filter for memory blocks.

See also

- EMemLeaks Unit 109

7.90 EMLang

MLang headers.

Remarks

This unit contains translation of mlang.h.

See also

- MLang

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.91 EModules

This is unit EModules

See also

- EModules Routines 114
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EHash Unit 97
- EHook Unit 99
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ENT Unit 114
- EPEImage Unit 115
- ESpecificDelphi Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

7.91.1 Routines

GetCompilationDate 408	Returns the compilation date of the specified executable module.

See also

- EModules Unit 114

7.92 EMonitors

This is unit EMonitors

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- ELowLevel Unit 105
- ETypes Unit 138

7.93 ENT

This is unit ENT

See also

- EDLLs Unit 78

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 105
- ETypes Unit 138

7.94 EOSApiList

This is unit `EOSApiList`

See also

- EConsts Unit 65
- EDLLs Unit 78
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EResLeaks Unit 116
- ETypes Unit 138

7.95 EPChars

This is unit `EPChars`

See also

- ETypes Unit 138

7.96 EPEImage

PE(64) formats support unit.

Remarks

This unit contains classes and routines to read and write Windows executable files (PE and PE64 formats).

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EHash Unit 97
- ELowLevel Unit 105
- ENT Unit 114
- ETools Unit 137
- ETypes Unit 138
- EUnmangling Unit 139

7.97 EPNG

This is unit `EPNG`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- ELowLevel Unit 105
- ETypes Unit 138
- EZLib Unit 145

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.98 EResLeaks

This is unit `EResLeaks`

See also

- EResLeaks Types 116
- EResLeaks Routines 116
- EConsts Unit 65
- EDisAsm Unit 78
- EDLLs Unit 78
- EHook Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ELowLevelClasses Unit 108
- ELowLevelExceptions Unit 108
- EMemLeaks Unit 109
- EOSApiList Unit 115
- EStackTracing Unit 133
- EStrConsts Unit 134
- ETypes Unit 138

7.98.1 Routines

HookWin32API 434	Adds a function to resource monitoring.

See also

- EResLeaks Unit 116

7.98.2 Types

TCheckType 619	Function check kind.

See also

- EResLeaks Unit 116

7.99 EResourceStrings

This is unit `EResourceStrings`

See also

- EHook Unit 99
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETypes Unit 138

7.100 ERunTime

This is unit `ERunTime`

See also

- EAppType Unit 54

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- EClassUtils Unit 61
- ECompatibility Unit 62
- EComponent Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDbgHelp Unit 65
- EDebugDWARF Unit 66
- EDebugEL Unit 66
- EDebugEL7 Unit 67
- EDebugExportBase Unit 67
- EDebugExports Unit 67
- EDebugInfo Unit 68
- EDebugJCL Unit 69
- EDebugMAB Unit 69
- EDebugMAB2 Unit 69
- EDebugMap Unit 70
- EDebugME Unit 70
- EDebugMS Unit 70
- EDebugStabs Unit 71
- EDebugTD32 Unit 71
- EDialog Unit 72
- EDialogCGI Unit 73
- EDialogConsole Unit 73
- EDialogHTMLBased Unit 74
- EDialogSend Unit 74
- EDialogSendWinAPI Unit 74
- EDialogService Unit 75
- EDialogWeb Unit 75
- EDialogWER Unit 75
- EDialogWinAPI Unit 76
- EDialogWinAPIEurekaLog Unit 76
- EDialogWinAPIEurekaLogDetailed Unit 76
- EDialogWinAPIMSClassic Unit 77
- EDialogWinAPIStepsToReproduce Unit 77
- EDisAsm Unit 78
- EDLLs Unit 78
- EDNS Unit 79
- EDWARF Unit 79
- EEncoding Unit 79
- EEncrypt Unit 80
- EEvents Unit 83
- EException Unit 89
- EExceptionHook Unit 90
- EExceptionInfo Unit 90
- EExceptionInfoBCB Unit 91
- EExceptionInfoCLR Unit 91
- EExceptionInfoCpp Unit 91
- EExceptionInfoDelphi2 Unit 92
- EExceptionInfoDelphiANSI Unit 92
- EExceptionInfoDelphiUnicode Unit 92
- EExceptionInfoGeneric Unit 92
- EExceptionInfoLLVM Unit 93
- EExceptionInfoMSVC Unit 93
- EExceptionManager Unit 93
- EFileMemory Unit 95
- EFonts Unit 95

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EFreeze Unit 96
- EHash Unit 97
- EHook Unit 99
- EHTMLHelp Unit 99
- EInfoFormat Unit 99
- EInject Unit 100
- EInterfaces Unit 100
- EJSON Unit 100
- EListView Unit 100
- ELogBuilder Unit 101
- ELogging Unit 103
- ELogManager Unit 105
- ELowLevel Unit 105
- ELowLevelClasses Unit 108
- ELowLevelExceptions Unit 108
- EMantisConnectStub Unit 108
- EMAPI Unit 108
- EMapParserBase Unit 109
- EMemLeaks Unit 109
- EMLang Unit 113
- EModules Unit 114
- EMonitors Unit 114
- ENT Unit 114
- EOSApiList Unit 115
- EPChars Unit 115
- EPEImage Unit 115
- EPNG Unit 115
- EResLeaks Unit 116
- EResourceStrings Unit 116
- EScreenShot Unit 119
- ESend Unit 120
- ESendAPIBugZilla Unit 121
- ESendAPIExceptionless Unit 122
- ESendAPIFogBugz Unit 122
- ESendAPIGitHub Unit 123
- ESendAPIGitLab Unit 124
- ESendAPIJIRA Unit 124
- ESendAPIJSON Unit 125
- ESendAPIMantis Unit 125
- ESendAPIRedmine Unit 126
- ESendAPIYouTrack Unit 126
- ESendMail Unit 127
- ESendMailMAPI Unit 127
- ESendMailShell Unit 128
- ESendMailSMAPI Unit 128
- ESendMailSMTP Unit 129
- ESendWeb Unit 129
- ESendWebFTP Unit 130
- ESendWebHTTP Unit 130
- ESignature Unit 131
- ESOAP Unit 132
- ESOAPMantis Unit 132
- ESocket Unit 132
- ESockets Unit 132
- ESpecificDelphi Unit 133
- ESSLSocket Unit 133
- EStackTracing Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETaskDialogs Unit 136
- EThreadsManager Unit 136

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ETools Unit 137
- ETypes Unit 138
- EUnmangling Unit 139
- EWCT Unit 139
- EWCTSupport Unit 140
- EWebTools Unit 140
- EWER Unit 140
- EWinCrypt Unit 141
- EWinSock Unit 141
- ExceptionLog7 Unit 141
- EXML Unit 143
- EXMLBuilder Unit 143
- EXMLRPC Unit 143
- EXMLRPCBugZilla Unit 144
- EZip Unit 144
- EZLib Unit 145
- EZLibHeaders Unit 145

7.101 EScreenShot

Screenshot support unit.

Remarks

This unit contains helper routines for taking screenshots.

See also

- Screenshot Routine 541
- ScreenshotInit Routine 543
- EScreenShot Types 120
- EScreenShot Routines 119
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EPNG Unit 115
- ESysInfo Unit 134
- ETypes Unit 138

7.101.1 Routines

FindScreenshotRect 400	Determinates screen area occupied by the specified object.
Screenshot 541	Takes screenshot.
ScreenshotFree 543	Release screenshot.
ScreenshotInit 543	Creates screenshot.
ScreenshotSave 544	Saves screenshot to file or stream.

See also

- EScreenShot Unit 119

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.101.2 Types

TScreeenshotHandle 721	Screenshot descriptor.

See also

- EScreenShot Unit 119

7.102 ESend

Base unit for send methods.

Remarks

This unit contains basic abstract class for implementing send methods in EurekaLog.

Actual send methods are implemented in other units (see below), which refers to this unit (ESend) andB implement one or more child sub-classes.

See also

- ESendAPIBugZilla 121
- ESendAPIFogBugz 122
- ESendAPIJIRA 124
- ESendAPIMantis 125
- ESendAPIRedmine 126
- ESendAPIYouTrack 126
- ESendAPIExceptionless 122
- ESendMailMAPI 127
- ESendMailShell 128
- ESendMailSMAPI 128
- ESendMailSMTP 129
- ESendWebFTP 130
- ESendWebHTTP 130
- EDialog 72
- EDialogSend 74
- ESend Classes 121
- ESend Types 121
- ESend Routines 121
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.102.1 Classes

TELUniversalSender 171	Base class for all send methods.

See also
- ESend Unit 120

7.102.2 Routines

EnumSenders 387	Returns list of registered send method class names.
FindSender 401	Searches send method class by name.
RegisterSender 520	Registers send method class.
RegisterSenderFirst 520	Registers send method class.

See also
- ESend Unit 120

7.102.3 Types

TELUniversalSenderClass 669	Pointer to send method class.

See also
- ESend Unit 120

7.103 ESendAPIBugZilla

This is unit `ESendAPIBugZilla`

See also
- ESendAPIBugZilla Classes 122
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EHash Unit 97
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EWebTools Unit 140
- EXMLRPCBugZilla Unit 144

7.103.1 Classes

TELTrackerBugZillaSender 168	Implements API access to BugZilla bug tracker

See also

- ESendAPIBugZilla Unit 121

7.104 ESendAPIExceptionless

This is unit `ESendAPIExceptionless`

See also

- ESendAPIExceptionless Classes 122
- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EException Unit 89
- EHash Unit 97
- EInfoFormat Unit 99
- EJSON Unit 100
- ELogBuilder Unit 101
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140
- EXML Unit 143

7.104.1 Classes

TELTrackerExceptionlessSender 168	Implements API access to Exceptionless bug tracker

See also

- ESendAPIExceptionless Unit 122

7.105 ESendAPIFogBugz

This is unit `ESendAPIFogBugz`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ESendAPIFogBugz Classes 123
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EEncoding Unit 79
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendWeb Unit 129
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETypes Unit 138
- EWebTools Unit 140
- EXMLBuilder Unit 143

7.105.1 Classes

TELTrackerFogBugzSender 45	Implements API access to Fogbugz bug tracker

See also

- ESendAPIFogBugz Unit 122

7.106 ESendAPIGitHub

This is unit `ESendAPIGitHub`

See also

- ESendAPIGitHub Classes 123
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendWeb Unit 129
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.106.1 Classes

TELTrackerGitHubSender 169	Implements API access to GitHub bug tracker

See also

- ESendAPIGitHub Unit 123

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.107 ESendAPIGitLab

This is unit ESendAPIGitLab

See also

- ESendAPIGitLab Classes 124
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.107.1 Classes

TELTrackerGitLabSender 170	Implements API access to GitLab bug tracker

See also

- ESendAPIGitLab Unit 124

7.108 ESendAPIJIRA

This is unit ESendAPIJIRA

See also

- ESendAPIJIRA Classes 124
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.108.1 Classes

TELTrackerJIRASender 170	Implements API access to JIRA bug

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	tracker

See also

- ESendAPIJIRA Unit 124

7.109 ESendAPIJSON

This is unit `ESendAPIJSON`

See also

- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EHash Unit 97
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESendWeb Unit 129
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.110 ESendAPIMantis

This is unit `ESendAPIMantis`

See also

- ESendAPIMantis Classes 125
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EExceptionHook Unit 90
- ELogging Unit 103
- ELowLevel Unit 105
- EMantisConnectStub Unit 108
- EModules Unit 114
- ESend Unit 120
- ESendWeb Unit 129
- ESpecificDelphi Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140
- EXML Unit 143
- EXMLRPC Unit 143

7.110.1 Classes

TELTrackerMantisSender 44	Implements API access to Mantis bug

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	tracker

See also

- ESendAPIMantis Unit 125

7.111 ESendAPIRedmine

This is unit `ESendAPIRedmine`

See also

- ESendAPIRedmine Classes 126
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EInfoFormat Unit 99
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.111.1 Classes

TELTrackerRedmineSender 171	Implements API access to Redmine bug tracker

See also

- ESendAPIRedmine Unit 126

7.112 ESendAPIYouTrack

This is unit `ESendAPIYouTrack`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EJSON Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendAPIJSON Unit 125
- ESendWeb Unit 129
- EStrConsts Unit 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140
- EXML Unit 143

7.113 ESendMail

This is unit `ESendMail`

See also

- EAppType Unit 54
- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- ESend Unit 120
- ESendMailShell Unit 128
- ESysInfo Unit 134
- ETypes Unit 138

7.114 ESendMailMAPI

This is unit `ESendMailMAPI`

See also

- ESendMailMAPI Classes 127
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMAPI Unit 108
- ESend Unit 120
- ESendMail Unit 127
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

7.114.1 Classes

TELMailMAPISender 42	Implements sending via full MAPI protocol.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ESendMailMAPI Unit 127

7.115 ESendMailShell

This is unit `ESendMailShell`

See also

- ESendMailShell Classes 128
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendMail Unit 127
- ESysInfo Unit 134
- ETypes Unit 138

7.115.1 Classes

TELMailShellSender 41	Implements sending via mailto: protocol

See also

- ESendMailShell Unit 128

7.116 ESendMailSMAPI

This is unit `ESendMailSMAPI`

See also

- ESendMailSMAPI Classes 129
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- EMAPI Unit 108
- ESend Unit 120
- ESendMail Unit 127
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.116.1 Classes

TELMailSimpleMAPISender 41	Implements sending via Simple MAPI

See also
- ESendMailSMAPI Unit 128

7.117 ESendMailSMTP

This is unit `ESendMailSMTP`

See also
- ESendMailSMTP Classes 129
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EDNS Unit 79
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- ESendMail Unit 127
- ESocket Unit 132
- ESockets Unit 132
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWinSock Unit 141

7.117.1 Classes

TELMailSMTPBaseSender 166	Base class for SMTP sending.
TELMailSMTPClientSender 43	Implements sending via SMTP client protocol.
TELMailSMTPSender 42	No description available
TELMailSMTPServerSender 42	Implements sending via SMTP server protocol.

See also
- ESendMailSMTP Unit 129

7.118 ESendWeb

This is unit `ESendWeb`

See also
- EBase Unit 57

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Unit 58
- EClasses Unit 61
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- EEvents Unit 83
- EException Unit 89
- EHash Unit 97
- EInfoFormat Unit 99
- ELogging Unit 103
- ELowLevel Unit 105
- ESend Unit 120
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWebTools Unit 140

7.119 ESendWebFTP

This is unit ESendWebFTP

See also

- ESendWebFTP Classes 130
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- ECore Unit 65
- EEncoding Unit 79
- ELowLevel Unit 105
- ESend Unit 120
- ESendWeb Unit 129
- ETypes Unit 138
- EWebTools Unit 140

7.119.1 Classes

TELFTPSender 43	Implements sending via FTP uploading.

See also

- ESendWebFTP Unit 130

7.120 ESendWebHTTP

This is unit ESendWebHTTP

See also

- ESendWebHTTP Classes 131
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EEncoding Unit 79
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ESend Unit 120
- ESendWeb Unit 129
- ETypes Unit 138
- EWebTools Unit 140

7.120.1 Classes

TELHTTPSender 43	Implements sending via HTTP posting.

See also

- ESendWebHTTP Unit 130

7.121 ESignature

Unit for working with digital signatures of executable files.

Remarks

This unit contains routines to exract information about digital signature of executable file.

If you are looking for code to sign/verify - see EEncrypt 80 unit.

See also

- SignatureGetInfo Function 558
- EEncrypt Unit 80
- EEncoding Unit 79
- EHash Unit 97
- ESignature Types 131
- ESignature Routines 131
- EBaseClasses Unit 58
- ECompatibility Unit 62
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETools Unit 137
- ETypes Unit 138
- EWinCrypt Unit 141

7.121.1 Routines

SignatureGetInfo 558	Returns information about digital signature and its certificate of an executable file.

See also

- ESignature Unit 131

7.121.2 Types

TCertificateInfo 613	Certificate information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TSignatureInfo 731	Information about digital signature of executable file.

See also

- ESignature Unit 131

7.122 ESOAP

This is unit ESOAP

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EInfoFormat Unit 99
- ELowLevel Unit 105
- ETypes Unit 138

7.123 ESOAPMantis

This is unit ESOAPMantis

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EResLeaks Unit 116
- ETools Unit 137
- ETypes Unit 138
- EXML Unit 143
- EXMLRPC Unit 143

7.124 ESocket

This is unit ESocket

See also

- ELowLevel Unit 105
- ETypes Unit 138
- EWinSock Unit 141

7.125 ESockets

This is unit ESockets

See also

- EAppType Unit 54
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EDLLs Unit 78
- EDNS Unit 79
- EEncoding Unit 79
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- ESocket Unit 132
- ESSLSocket Unit 133
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWinCrypt Unit 141
- EWinSock Unit 141

7.126 ESpecificDelphi

This is unit `ESpecificDelphi`

See also

- EDLLs Unit 78
- ELowLevel Unit 105
- ETypes Unit 138

7.127 ESSLSocket

This is unit `ESSLSocket`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDLLs Unit 78
- ELowLevel Unit 105
- ESocket Unit 132
- ESysInfo Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWinCrypt Unit 141
- EWinSock Unit 141

7.128 EStackTracing

Low-level stack tracing code.

Remarks

This unit contains actual code that performs stack tracing. RAW results of this code will be processed by classes from ECallStack 59. Do not use code from this unit directly - always use `ECallStack` unit.

See also

- ECallStack unit 59
- EConsts Unit 65
- EDisAsm Unit 78
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.129 EStrConsts

String constants.

Remarks

This unit holds global localizable string constants for EurekaLog. This unit can be localized by 3rd party tools.

There are resourcestrings, as well as run-time arrays of string and OnTranslate (GetText-style) function. Your localization solution may use any approach. For example, it may localize resourcestrings. Or it may install OnTranslate handler.

See also

- EConsts Unit 65
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109

7.130 ESysInfo

System information functions.

Remarks

This unit contains information extraction routines, which are used by bug report generation code to create error reports.

Most routines from this unit can work in your application without EurekaLog enabled or installed.

See also

- Info formatting routines 99
- Log builder 101
- ESysInfo Routines 134
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- EHash Unit 97
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- ENT Unit 114
- EResLeaks Unit 116
- EStrConsts Unit 134
- ETools Unit 137
- ETypes Unit 138
- EWCTSupport Unit 140
- EWinSock Unit 141

7.130.1 Routines

ExpandEnvVars 398	Expands environment-variable strings.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

GetActiveFormClass 402	Gets window class of active window.
GetActiveFormText 403	Gets caption of active window.
GetApplicationDescription 404	Gets application description.
GetApplicationName 404	Gets application name.
GetApplicationRect 405	Gets bounding rectangle for GUI application.
GetCompanyName 407	Gets company name.
GetComputerName 409	Gets computer name user name.
GetControlClass 409	Gets class of active control.
GetControlText 410	Gets text of active control.
GetCurrentModuleCompilationDate 411	Gets compilation time of current module.
GetCurrentUserPrivileges 411	Gets description of user privileges.
GetDiskInfo 412	Gets total and free disk space size.
GetDisplayDPI 412	Gets screen DPI.
GetDisplayMode 413	Gets screen mode.
GetEnvVariable 413	Expands single environment variable.
GetEurekaLogLabel 414	Gets full EurekaLog version.
GetEurekaLogLicense 415	Gets EurekaLog license.
GetEurekaLogLicenseType 416	Gets EurekaLog license type.
GetEurekaLogPath 417	Obsolete function, do not use.
GetEurekaLogStatus 417	Gets EurekaLog stable/beta information.
GetEurekaLogVersion 418	Gets EurekaLog version.
GetExceptionModuleName 419	No description available
GetExceptionModuleVersion 419	No description available
GetFileIcon 420	No description available

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

GetFileSize 421	Gets file size.
GetUserEMail 432	No description available
IsWin10 471	Detect Windows 10 (minimum).
IsWin11 472	Detect Windows 11 (minimum).
IsWin64 472	Detect OS bitness.
IsWin7 473	Detect Windows 7 (minimum).
IsWin8 474	Detect Windows 8 (minimum).
IsWin81 474	Detect Windows 8.1 (minimum).
IsWinVista 475	Detect Windows Vista (minimum).
IsWinXP 476	Detect Windows XP (minimum).
IsWorkstation 476	Gets server/workstation status of current computer.
SetUserEmail 553	No description available

See also

- ESysInfo Unit 134

7.131 ETaskDialogs

This is unit `ETaskDialogs`

See also

- ETypes Unit 138

7.132 EThreadsManager

EurekaLog thread manager unit.

Remarks

This unit contains EurekaLog thread manager. EurekaLog uses thread manager to store per-thread information.

See also

- TEurekaThreadData Class 219
- ThreadData Function 577
- ThreadDataEnum Function 578
- EThreadsManager Classes 137
- EThreadsManager Types 137
- EThreadsManager Routines 137
- EAppType Unit 54
- EBase Unit 57

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EExceptionHook Unit 90
- EHook Unit 99
- EInject Unit 100
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ENT Unit 114
- EResLeaks Unit 116
- ESpecificDelphi Unit 133
- ETypes Unit 138

7.132.1 Classes

TEurekaThreadData 219	Thread information.
TEurekaThreadDataList 229	List of threads.

See also
- EThreadsManager Unit 136

7.132.2 Routines

AcquireThreadData 318	Returns thread information (any thread).
ThreadData 577	Returns thread information (current thread only).
ThreadDataEnum 578	Enumerates running threads.

See also
- EThreadsManager Unit 136

7.132.3 Types

TThreadKind 738	Indicates thread kind.

See also
- EThreadsManager Unit 136

7.133 ETools

This is unit `ETools`

See also
- EBaseClasses Unit 58
- ECompatibility Unit 62

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EConfig Unit 63
- EConsts Unit 65
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EStrConsts Unit 134
- ETypes Unit 138

7.134 ETypes

Basic EurekaLog types.

Remarks

This unit holds declarations of basic types and constants.

Tip
This unit is also `interface` counterpart to ECompatibility 62 unit. That's why this unit must be listed as last unit in `uses` clause of `interface` section.

See also

- ECompatibility 62
- ETypes Types 138

7.134.1 Types

PCustomName 607	Defines buffer for storing name of memory block.
PPointerRecord 608	A pointer to TPointerRecord type.
PtrUInt 609	Unsigned integer type with same size as `Pointer`.
TAppType 610	Defines application type.
TBlockType 611	Defines type of memory block.
TBugID 613	BugID type.
TCustomName 622	Defines buffer for storing name of memory block.
TEurekaActionType 672	Action kind for exception processing.
TEurekaDebugDetail 678	Level of detalization.
TEurekaDebugDetails 680	Set of detalization levels.
TEurekaLogFileType 680	Type of file name.
TEurekaModuleType 683	Alias for TModuleType 709

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TIssueIDType 690	Type of the issue ID in the bug tracker.
TMemoryError 705	Type of detected memory error.
TMemoryErrors 708	Defines set of possible memory errors.
TModuleType 709	Indicates kind of executable module.
TPointerRecord 710	Header for memory blocks allocated by EurekaLog.
TResponse 713	No description available
TSendResult 722	Result kind for exception processing.
TSendScreenshot 727	Specifies screen area.
TSysErrorCode 737	A type for GetLastError/SetLastError.

See also

- ETypes Unit 138

7.135 EUnmangling

This is unit `EUnmangling`

See also

- EBase Unit 57
- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- ELowLevel Unit 105
- EPEImage Unit 115
- ETools Unit 137
- ETypes Unit 138

7.136 EWCT

WCT headers.

Remarks

This unit contains translation of Windows headers for WCT (Wait Chain Traversal). It includes wct.h.

Call WCTInit 600 function before using this unit. (Optionally) Call WCTDone 600 function after using this unit.

See also

- EWCT Routines 140
- EDLLs Unit 78
- ELowLevel Unit 105

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

7.136.1 Routines

WCTDone 600	Finalize EWCT unit.
WCTInit 600	Initialize EWCT unit.

See also

- EWCT Unit 139

7.137 EWCTSupport

This is unit `EWCTSupport`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EDLLs Unit 78
- ELogging Unit 103
- ELowLevel Unit 105
- EMemLeaks Unit 109
- ETypes Unit 138
- EWCT Unit 139

7.138 EWebTools

This is unit `EWebTools`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EConsts Unit 65
- EDLLs Unit 78
- EEncoding Unit 79
- ELogging Unit 103
- ELowLevel Unit 105
- EModules Unit 114
- ESend Unit 120
- EStrConsts Unit 134
- ESysInfo Unit 134
- ETypes Unit 138

7.139 EWER

WER headers.

Remarks

This unit contains translation of C++ Headers for WER (Windows Error Reporting). Corresponding .h files are errorrep.h and werapi.h.

Call InitWER 440 function before using this unit.

See also

- InitWER 440
- ReportFault

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- WerReportSubmit
- EWER Routines 141
- EDLLs Unit 78
- ELowLevel Unit 105
- ETypes Unit 138

7.139.1 Routines

InitWER 440	Initializes EWER unit.

See also
- EWER Unit 140

7.140 EWinCrypt

Windows cryptography headers.

Remarks
This unit contains translation of Windows headers for WinCrypt. It includes schannel.h, sspi.h, wincrypt.h.

All functions in this unit are automatically imported on request. There is no need to call any initialization routine.

7.141 EWinSock

WinSock headers.

Remarks
This unit contains translation of Windows headers for WinSock. It includes winsock.h, winsock2.h, ws2tcpip.h.

All functions in this unit are automatically imported on request. There is no need to call any initialization routine.

See also
- EDLLs Unit 78
- ELowLevel Unit 105

7.142 ExceptionLog7

Main EurekaLog 7 unit.

Remarks
This unit contains control logic of EurekaLog. It must be included into any application which use EurekaLog.

This unit also includes common routines of EurekaLog.

Note
Consider using EBase 57 unit if you want to call EurekaLog routines in application without EurekaLog enabled (such as IsEurekaLogActive 452). Including `ExceptionLog7` unit into your application will include EurekaLog's code - which is not needed for applications without EurekaLog enabled. Including `EBase` unit will include only minimum stub code without full overhead of entire EurekaLog's

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

code.

Note

You can check whenever `ExceptionLog7` unit is included into executable module by using PE Informer (aka Module Analyzer) Tool, which is included with EurekaLog 7 or EurekaLog Tools Pack.

See also

- ExceptionLog7 Routines 142
- EAppType Unit 54
- EBase Unit 57
- EBaseClasses Unit 58
- ECallStack Unit 59
- EClasses Unit 61
- EClassUtils Unit 61
- ECompatibility Unit 62
- EConfig Unit 63
- EConsts Unit 65
- ECore Unit 65
- EDebugEL Unit 66
- EDebugInfo Unit 68
- EDialog Unit 72
- EDisAsm Unit 78
- EDLLs Unit 78
- EEncoding Unit 79
- EEncrypt Unit 80
- EEvents Unit 83
- EException Unit 89
- EExceptionHook Unit 90
- EExceptionInfo Unit 90
- EExceptionManager Unit 93
- EFreeze Unit 96
- EHook Unit 99
- EInfoFormat Unit 99
- EInject Unit 100
- ELogBuilder Unit 101
- ELogging Unit 103
- ELogManager Unit 105
- ELowLevel Unit 105
- EMemLeaks Unit 109
- EModules Unit 114
- ENT Unit 114
- EPEImage Unit 115
- EResLeaks Unit 116
- ESend Unit 120
- ESendMail Unit 127
- ESendMailShell Unit 128
- ESpecificDelphi Unit 133
- EStackTracing Unit 133
- EStrConsts Unit 134
- ESysInfo Unit 134
- EThreadsManager Unit 136
- ETypes Unit 138

7.142.1 Routines

CurrentEurekaLogOptions 8	No description available

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IsEurekaLogActive 3	Indicates if EurekaLog is active or disabled in this application.
IsEurekaLogActiveInThread 5	Indicates if EurekaLog is active or disabled in the specified thread.
IsEurekaLogInstalled 2	Indicates if EurekaLog is installed in this application.
LastBugID 9	Returns BugID of last exception in current thread.
SetEurekaLogState 6	Enables or disables EurekaLog in this application.
SetEurekaLogStateInThread 7	Enables or disables EurekaLog in the specified thread.

See also

- ExceptionLog7 Unit 141

7.143 EXML

This is unit EXML

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELowLevel Unit 105
- ETools Unit 137
- ETypes Unit 138

7.144 EXMLBuilder

This is unit EXMLBuilder

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELowLevel Unit 105
- ETools Unit 137
- ETypes Unit 138

7.145 EXMLRPC

This is unit EXMLRPC

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EEncoding Unit 79
- ELowLevel Unit 105
- ETypes Unit 138
- EWebTools Unit 140
- EXML Unit 143

7.146 EXMLRPCBugZilla

This is unit `EXMLRPCBugZilla`

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ECore Unit 65
- EEncoding Unit 79
- EInfoFormat Unit 99
- ELowLevel Unit 105
- ETools Unit 137
- ETypes Unit 138
- EXML Unit 143
- EXMLRPC Unit 143

7.147 EZip

Compression support unit.

Remarks

This unit is hi-level wrapper for ZLib. It contains routines to compress/decompress RAW binary data in memory, as well as tools to pack/unpack files into ZIP archives.

Password and UTF-8 support is present.

See also

- EZLib 145
- EZip Classes 144
- EZip Types 145
- EZip Routines 145
- EZip Globals 144
- EBaseClasses Unit 58
- ETypes Unit 138

7.147.1 Classes

EZIPException 151	Exception type for all Zip errors.
TZipFile 266	Class for creating and reading .ZIP files.

See also

- EZip Unit 144

7.147.2 Globals

SIGNATURE_CENTRALHEADER 757	Signature of .zip file.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

SIGNATURE_DATADESCRIPTOR 758	Data descriptor signature.
SIGNATURE_LOCALHEADER 758	Local file header signature.
SIGNATURE_ZIPENDOFHEADER 759	End of central dir signature.

See also
- EZip Unit 144

7.147.3 Routines

TZipCompressionToString 586	Converts ZIP compression method value to string representation.

See also
- EZip Unit 144

7.147.4 Types

TStreamConstructor 736	Function to Create a Compression/ Decompression stream
TZipCompression 741	Zip compression method.
TZipEndOfCentralHeader 747	Final block written to zip file
TZipHeader 747	TZipHeader contains information about a file in a zip archive.

See also
- EZip Unit 144

7.148 EZLib

Compression support unit.

Remarks
This is a header unit for ZLib. ZLib implementation is already compiled by C++ compiler into a set of .obj files. No external DLL used.

See also
- EZip 144

7.149 EZLibHeaders

This is unit `EZLibHeaders`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- EBaseClasses Unit 58
- ECompatibility Unit 62
- ETypes Unit 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part VII

8 All Classes

EDNSError 151	Base class for DNS errors.
EZIPException 151	Exception type for all Zip errors.
TBaseDialog 36	Base class for all dialogs.
TBaseLogBuilder 152	Base class for log builder.
TConsoleWriterDialog 37	No description available
TDummyLogBuilder 159	Dummy log builder class.
TELAbstractMapParser 159	Base class for .map file parsers.
TELDebugInfoExports 39	No description available
TELDebugInfoJedi 39	No description available
TELDebugInfoMadExcept 39	No description available
TELDebugInfoMap 39	No description available
TELDebugInfoNative 39	No description available
TELDebugInfoSource 164	Base class for debug information provider.
TELDebugInfoSymbols 39	No description available
TELDebugInfoTD32 39	No description available
TELFTPSender 43	Implements sending via FTP uploading.
TELHTTPSender 43	Implements sending via HTTP posting.
TELMailMAPISender 42	Implements sending via full MAPI protocol.
TELMailShellSender 41	Implements sending via mailto: protocol
TELMailSimpleMAPISender 41	Implements sending via Simple MAPI
TELMailSMTPBaseSender 166	Base class for SMTP sending.
TELMailSMTPClientSender 43	Implements sending via SMTP client protocol.
TELMailSMTPSender 42	No description available

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TELMailSMTPServerSender 42	Implements sending via SMTP server protocol.
TELTrackerBugZillaSender 44	No description available
TELTrackerBugZillaSender 168	Implements API access to BugZilla bug tracker
TELTrackerExceptionlessSender 168	Implements API access to Exceptionless bug tracker
TELTrackerFogBugzSender 45	Implements API access to Fogbugz bug tracker
TELTrackerFogBugzSender 44	No description available
TELTrackerGitHubSender 169	Implements API access to GitHub bug tracker
TELTrackerGitLabSender 170	Implements API access to GitLab bug tracker
TELTrackerJIRASender 170	Implements API access to JIRA bug tracker
TELTrackerMantisSender 44	Implements API access to Mantis bug tracker
TELTrackerMantisSender 44	No description available
TELTrackerRedmineSender 171	Implements API access to Redmine bug tracker
TELUniversalSender 171	Base class for all send methods.
TEurekaBaseStackList 172	Base call stack class.
TEurekaExceptionInfo 180	Exception information class.
TEurekaLogDetailedDialog 37	No description available
TEurekaLogDialog 37	No description available
TEurekaLogEvents 33	EurekaLog 7 component.
TEurekaModuleOptions 216	EurekaLog options.
TEurekaStackListV6 217	Stack tracing class for EurekaLog 6 method.
TEurekaStackListV7 218	Stack tracing class for EurekaLog 7 RAW

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	method.
TEurekaThreadData 219	Thread information.
TEurekaThreadDataList 229	List of threads.
TEventLogDialog 37	No description available
TExceptionManager 232	Exception manager class.
TFreezeThread 246	Base class for freeze detection threads.
TGenericStackList 249	Common stack tracing class.
TJSONLogBuilder 252	JSON EurekaLog 7+ log builder.
TLogBuilder 253	Default EurekaLog 7 log builder.
TMainThreadFreezeDetectionThread 254	Thread class to detects hangs in UI thread.
TMessageBoxDetailedDialog 37	No description available
TMessageBoxDialog 37	No description available
TMSClassicDialog 37	No description available
TNullDialog 37	No description available
TRTLHandlerDialog 37	No description available
TThreadEx 255	Abstract class to create separate threads of execution.
TWCTFreezeDetectionThread 265	Thread class to detects deadlocks between threads.
TWebDialog 37	No description available
TWebDialog 37	No description available
TWebDialog 37	No description available
TWERDialog 37	No description available
TXMLLogBuilder 265	XML EurekaLog 7+ log builder.
TZipFile 266	Class for creating and reading .ZIP files.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.1 EDNSError

Base class for DNS errors.

Unit
EDNS 79

Syntax

Code (Delphi)
```
EDNSError = class(EEurekaLogGeneralError);
```

Remarks

See also
- GetMXServerFromEmail 426

8.2 EZIPException

Exception type for all Zip errors.

Unit
EZip 144

Syntax

Code (Delphi)
```
EZIPException = class(Exception);
```

Remarks

8.3 TBaseDialog

Base class for all dialogs.

Unit
EDialog 72

Syntax

Code (Delphi)
```
TBaseDialog = class(TObject);
```

Remarks
This is base abstract class for error dialogs in EurekaLog. It provides service methods and implements common interface.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Actual (non abstract) dialogs are implemented as child sub-classes for this class. Each dialog class must be registered via RegisterDialogClass 502 or RegisterDialogClassFirst 503 functions.

Each child class must implement at least ShowModalInternal method (protected), where it must implement functionality of the dialog.

8.4 TBaseLogBuilder

Base class for log builder.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TBaseLogBuilder = class(TObject);
```

Remarks

This is abstract class for any specific log builder. Log builder is a class which is used by EurekaLog to create bug reports from exception information. There is a specific sub-class inherited from this abstract class which is used by default: TLogBuilder 253.

Another build-in log builder classes are TXMLLogBuilder 265, TJSONLogBuilder 252 and TDummyLogBuilder 159.

Any other log builder class should be defined by user. Any specific sub-class must implement `BuildReport` method.

Normally you don't need to create instances of this class manually. See BuildBugReport 328.

Note

Bug report is a simple multi-line text. Multi-files bug reports (ELP/ZIP) are created by dialog class 36.

Methods 152	Methods
Properties 155	Properties
protected	Protected members
public	Public members

8.4.1 Methods

Create 153	Creates instance of log builder class.
BuildReport 153	Builds bug report.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Refresh 154	Refreshes (i.e. recreates) bug report.
ToString 155	Returns textual representation of class instance.

See also

- TBaseLogBuilder 152
- ELogBuilder 101

8.4.1.1 Create

Creates instance of log builder class.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
public
constructor Create(
   const AExceptionInfo: TEurekaExceptionInfo 180
); virtual;
```

Parameters

AExceptionInfo [in]

Exception information to build bug report. This information is saved to TBaseLogBuilder.ExceptionInfo 157 property.

Remarks

This is default constructor for any log builder class.

Note

Creating log builder instance will not generate bug report. Bug report will not be created until TBaseLogBuilder.Report 158 property is accessed. I.e. bug report is created on demand.

See also

- CustomFields property 157
- ExceptionInfo property 157
- Options property 158
- Report property 158
- BuildReport method 153
- ToString method 155
- TBaseLogBuilder Class 152

8.4.1.2 BuildReport

Builds bug report.

Unit

ELogBuilder 101

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
protected
function BuildReport: String;
   virtual;
   abstract;
```

Return value

Textual representation of bug report.

Remarks

This is abstract method which generates actual bug report from exception information. Exception information can be obtained via TBaseLogBuilder.ExceptionInfo 157 property. TBaseLogBuilder also has some other useful properties for bug report generation.

Any non-abstract sub-class of TBaseLogBuilder class should implement this method.

Caution!

Do not call this method directly. Call TBaseLogBuilder.ToString 155 method or TBaseLogBuilder.Report 158 property instead.

Note

Usually this method is called only once per class instance. TBaseLogBuilder uses caching of bug report for performance reasons. See also: TBaseLogBuilder.Refresh 154

See also

- Refresh method 154
- FBuilded field 156
- Options property 158
- CallStack property 156
- ExceptionInfo property 157
- CustomFields property 157
- ToString method 155
- ESysInfo unit 134
- TBaseLogBuilder Class 152

8.4.1.3 Refresh

Refreshes (i.e. recreates) bug report.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
protected
procedure Refresh;
   virtual;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Sets TBaseLogBuilder.FBuilded 156 to False, so next access to TBaseLogBuilder.Report 158 property will generate new bug report. Use this method to reflect changes in exception or environment information (for example, user e-mail was changed in dialog).

See also

- FBuilded field 156
- BuildReport method 153
- Report property 158
- ToString method 155
- TBaseLogBuilder Class 152

8.4.1.4 ToString

Returns textual representation of class instance.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
public
function ToString: String;
   override;
```

Return value

Value of TBaseLogBuilder.Report 158 property.

Remarks

Returns bug report. Calling this method is the same as accessing TBaseLogBuilder.Report 158 property.

See also

- Report property 158
- BuildReport method 153
- TBaseLogBuilder Class 152

8.4.2 Properties

FBuilded 156	Specifies if bug report was created or not.
CallStack 156	Call stack.
CustomFields 157	Custom fields for `Options.soCustomData`.
ExceptionInfo 157	Exception information.
Options 158	EurekaLog options.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Report 158	Generates and returns bug report.

See also

- TBaseLogBuilder 152
- ELogBuilder 101

8.4.2.1 FBuilded

Specifies if bug report was created or not.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
protected
FBuilded: Boolean;
```

Value

`True`: bug report was created, TBaseLogBuilder.Report 158 property will return cached created bug report from internal field. `False`: bug report was not created, TBaseLogBuilder.Report 158 property will generate new report and cache it into internal field.

Remarks

This flag is used by caching. Bug report is not created each time TBaseLogBuilder.Report 158 property is accessed for performance reasons. If you want to explicitly regenerate bug report (for example, to reflect latest changes in exception information or environment) - use TBaseLogBuilder.Refresh 154 method.

Note

Caching is performed automatically. You don't need to do anything special in TBaseLogBuilder.BuildReport 153 method. TBaseLogBuilder class will automatically cache bug report for later use.

See also

- TBaseLogBuilder Class 152

8.4.2.2 CallStack

Call stack.

Unit

ELogBuilder 101

Value

Call stack of exception. Default is TBaseLogBuilder.ExceptionInfo 157.CallStack 188.

Remarks

This property can be used by TBaseLogBuilder.BuildReport 153 method. This property can be overriden by sub-classes to customize bug report content without affecting exception information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ExceptionInfo property 157
- TEurekaExceptionInfo class 180
- BuildReport method 153
- TBaseLogBuilder Class 152

8.4.2.3 CustomFields

Custom fields for `Options.soCustomData`.

Unit

ELogBuilder 101

Value

Set of name-value pairs to be used as custom data in bug reports.

Remarks

This property can be used to specify custom data for bug report. Typically you fill this property right after creation of log builder instance and before accessing TBaseLogBuilder.Report 158 property.

Tip
Name-value pairs can be easily specified by using TStrings.Values property.

Note
Altering `CustomFields` property will not refresh bug report automatically. You need to call TBaseLogBuilder.Refresh 154 method to reflect changes.

See also

- BuildReport method 153
- Refresh method 154
- TBaseLogBuilder Class 152

8.4.2.4 ExceptionInfo

Exception information.

Unit

ELogBuilder 101

Value

Exception information to build bug report.

Remarks

This is exception information object which was passed to constructor 153. TBaseLogBuilder.BuildReport 153 method can use this property to gather information about exception.

Note
There are TBaseLogBuilder.CallStack 156 and TBaseLogBuilder.Options 158 properties which should be used instead of TBaseLogBuilder.ExceptionInfo.CallStack 188 and TBaseLogBuilder.ExceptionInfo.Options 201 property. TBaseLogBuilder.CallStack 156 and TBaseLogBuilder.Options 158 properties can be overriden by sub-classes to customize bug report content without affecting exception information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEurekaExceptionInfo class 180
- Options property 158
- CallStack property 156
- Report property 158
- BuildReport method 153
- TBaseLogBuilder Class 152

8.4.2.5 Options

EurekaLog options.

Unit

ELogBuilder 101

Value

Call stack of exception. Default is TBaseLogBuilder.ExceptionInfo 157.Options 201.

Remarks

This property can be used by TBaseLogBuilder.BuildReport 153 method. This property can be overriden by sub-classes to customize bug report content without affecting exception information.

See also

- ExceptionInfo property 157
- TEurekaExceptionInfo class 180
- BuildReport method 153
- TBaseLogBuilder Class 152

8.4.2.6 Report

Generates and returns bug report.

Unit

ELogBuilder 101

Value

Textual representation of bug report.

Remarks

Call this property to generate and retrieve bug report for exception information which was specified in constructor 153 of log builder instance. Specified exception information can be checked by reading TBaseLogBuilder.ExceptionInfo 157 property.

Note
Actual bug report generation takes place in TBaseLogBuilder.BuildReport 153 method.

See also

- ToString method 155
- BuildReport method 153
- Refresh method 154
- TBaseLogBuilder Class 152

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.5 TConsoleWriterDialog

Enter topic text here.

8.6 TDummyLogBuilder

Dummy log builder class.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TDummyLogBuilder = class(TBaseLogBuilder 152);
```

Remarks

This log builder class can "generate" a pre-made bug report. You can pass arbitrary string to constructor of this class and this string will be returned as bug report. This class can not be used as default log builder class (i.e. via LogBuilderClass 757) - because there is no way to pass "report string" to alternative constructor), but it is usefull to use already generated bug report in dialogs.

8.7 TELAbstractMapParser

Base class for .map file parsers.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

```
TELAbstractMapParser = class(TObject);
```

Remarks

Methods 159	Methods
protected	Protected members
public	Public members

8.7.1 Methods

DecodeIdent 160	Returns humanreadable string from ident, returned by one of

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	MapStringToXYZ functions.
MapStringToFileName 160	Reads file name from mapped image of .map file.
MapStringToModuleName 161	Reads unit name from mapped image of .map file.
MapStringToStr 162	Reads string from mapped image of .map-file.
PrepareStr 163	Converts string from .map file to Delphi string.

See also

- TELAbstractMapParser 159
- EMapParserBase 109

8.7.1.1 DecodeIdent

Returns humanreadable string from ident, returned by one of MapStringToXYZ functions.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

```
public
class function DecodeIdent(
   const AIdent: String
): String;
```

Parameters

AIdent

Result of any MapStringToXYZ method.

See also

- TELAbstractMapParser Class 159

8.7.1.2 MapStringToFileName

Reads file name from mapped image of .map file.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
public
class function MapStringToFileName(
   AMapString: PELMapString
): String; virtual;
```

Parameters

AMapString

A pointer to the beginning of the string to read. Usually points to '(' (see MapStringToStr).

Remarks

Reads file name from the "Line numbers for Unit1(code.inc) segment .text" section of .map file. See examples.

This method behaves differently on different Delphi versions:

- ANSI version of this method returns UTF-8 encoded string.
- Unicode version of this method returns UTF-16 encoded string.

Examples

Code (Delphi)

```
// Reads full file name:
S := MapStringToModuleName('(C:\Documents and Settings\user\Мои документ
// S = 'C:\Documents and Settings\user\Мои документы\RAD Studio\Projects
//
// Reads short file name:
S := MapStringToModuleName('code.inc) segment .text');
// S = 'code.inc';
```

See also

- TELAbstractMapParser.MapStringToModuleName
- TELAbstractMapParser.MapStringToStr
- TMapFileParser.MapStringToIdent
- TELAbstractMapParser.PrepareStr
- TELAbstractMapParser Class 159

8.7.1.3 MapStringToModuleName

Reads unit name from mapped image of .map file.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

```
public
class function MapStringToModuleName(
   AMapString: PELMapString
): String; virtual;
```

Parameters

AMapString

A pointer to the beginning of the string to read.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Reads unit name from the "Detailed map of segments" section of .map file. This function strips library name (C ++ Builder only) and '.pas' and '.obj' extensions. See examples.

This method behaves differently on different Delphi versions:
- ANSI version of this method returns UTF-8 encoded string.
- Unicode version of this method returns UTF-16 encoded string.

Examples

Code (Delphi)

```
// Reads name of Delphi unit:
S := MapStringToModuleName('TypInfo ACBP=A9');
// S = 'TypeInfo'
//
// Reads name of C++ Builder unit:
S := MapStringToModuleName('C:\PROGRAM FILES\BORLAND\CBUILDER6\LIB\RELEA
// S = 'ShlObj';
```

See also

- TELAbstractMapParser.MapStringToStr
- TELAbstractMapParser.MapStringToFileName
- TMapFileParser.MapStringToIdent
- TELAbstractMapParser.PrepareStr
- TELAbstractMapParser Class 159

8.7.1.4 MapStringToStr

Reads string from mapped image of .map-file.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

```
protected
class function MapStringToStr(
   AMapString: PELMapString;
   const AStopOnSpace, AHasRVAColumn: Boolean
): String; virtual;
```

Parameters

AMapString

A pointer to the beginning of the string to read.

Remarks

If `AMapString` starts with '(' - read string between '()', return data inside '()'.If `AMapString` starts with any other character: read string until EOL. This method behaves differently on different Delphi versions: ANSI version of this method returns UTF-8 encoded string.Unicode version of this method returns UTF-16 encoded string. Use TELAbstractMapParser.DecodeIdentDecodeIdent to get readable string.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
// Reads until EOL:
S := MapStringToStr('SysConst ACBP=A9'#13#10' 0002:000014F0 00000114 C=E
// S = 'SysConst'
//
// Reads until EOL:
S := MapStringToStr('AnsiLowerCase'#13#10' 0001:00008158 AnsiLowerCaseFi
// S = 'AnsiLowerCase'
//
// Reads until EOL:
S := MapStringToStr('__fastcall System::AnsiString::operator =(const Sys
// S = '__fastcall System::AnsiString::operator =(const System::AnsiStri
```

See also

- TELAbstractMapParser.MapStringToModuleName
- TELAbstractMapParser.MapStringToFileName
- TMapFileParser.MapStringToIdent
- TELAbstractMapParser.PrepareStr
- TELAbstractMapParser Class 159

8.7.1.5 PrepareStr

Converts string from .map file to Delphi string.

Unit

EMapParserBase 109

Syntax

Code (Delphi)

```
protected
class function PrepareStr(
   AStart: PELMapString;
   ALen: Integer
): String; virtual;
```

Parameters

AStart

Pointer to the begginning of the string.

ALen

Length of string in characters.

Remarks

ANSI version of this method returns UTF-8 encoded string.

Unicode version of this method returns UTF-16 encoded string.

This is low-level function to convert string from map file to proper encoding. Other routines (MapStringToXYZ) uses this function to prepare result before return.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Use DecodeIdent to get readable string.

See also

- TELAbstractMapParser.DecodeIdent
- TELAbstractMapParser.MapStringToStr
- TELAbstractMapParser.MapStringToModuleName
- TELAbstractMapParser.MapStringToFileName
- TMapFileParser.MapStringToIdent
- TELAbstractMapParser Class 159

8.8 TELDebugInfoExports

Enter topic text here.

8.9 TELDebugInfoJedi

Enter topic text here.

8.10 TELDebugInfoMadExcept

Enter topic text here.

8.11 TELDebugInfoMap

Enter topic text here.

8.12 TELDebugInfoNative

Enter topic text here.

8.13 TELDebugInfoSource

Base class for debug information provider.

Unit

EBaseClasses 58

Syntax

Code (Delphi)

```
TELDebugInfoSource = class(TCheckedInterfacedObject);
```

Remarks

This is the base class for debug information providers. Debug information provider is a class which can load and parse debug information in one specific format. Debug info provider can be registered in EDebugInfo unit 68

Child classes must override at least InitializeSource and GetLocationInfo methods. Most providers will also override Heuristic method.

8.14 TELDebugInfoSymbols

Enter topic text here.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.15 TELDebugInfoTD32

Enter topic text here.

8.16 TELFTPSender

Implements sending via FTP uploading.

Unit
ESendWebFTP 130

Syntax

Code (Delphi)
```
TELFTPSender = class(TELWebFilesSender);
```

Remarks
Application settings must be set up so each bug report has unique file name.

8.17 TELHTTPSender

Implements sending via HTTP posting.

Unit
ESendWebHTTP 130

Syntax

Code (Delphi)
```
TELHTTPSender = class(TELWebFilesSender);
```

Remarks
Requires a custom receiver script or form.

8.18 TELMailMAPISender

Implements sending via full MAPI protocol.

Unit
ESendMailMAPI 127

Syntax

Code (Delphi)
```
TELMailMAPISender = class(TELMailSender);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks
Implemented by Outlook.

8.19 TELMailShellSender

Implements sending via mailto: protocol

Unit
ESendMailShell 128

Syntax

```
Code (Delphi)
TELMailShellSender = class(TELMailSender);
```

Remarks
Works via ShellExecute.

8.20 TELMailSimpleMAPISender

Implements sending via Simple MAPI

Unit
ESendMailSMAPI 128

Syntax

```
Code (Delphi)
TELMailSimpleMAPISender = class(TELMailSender);
```

Remarks
Works via SMTP stub, can work in cross-bitness.

8.21 TELMailSMTPBaseSender

Base class for SMTP sending.

Unit
ESendMailSMTP 129

Syntax

```
Code (Delphi)
TELMailSMTPBaseSender = class(TELMailSender);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

8.22 TELMailSMTPClientSender

Implements sending via SMTP client protocol.

Unit
ESendMailSMTP 129

Syntax

Code (Delphi)

```
TELMailSMTPClientSender = class(TELMailSMTPBaseSender 166);
```

Remarks
Same as normal e-mail client application.

8.23 TELMailSMTPSender

Unit
ESendMailSMTP 129

Syntax

Code (Delphi)

```
TELMailSMTPSender = class(TELMailSMTPBaseSender 166);
```

Remarks

8.24 TELMailSMTPServerSender

Implements sending via SMTP server protocol.

Unit
ESendMailSMTP 129

Syntax

Code (Delphi)

```
TELMailSMTPServerSender = class(TELMailSMTPBaseSender 166);
```

Remarks
Works by creating a fake e-mail server.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.25 TELTrackerBugZillaSender

Enter topic text here.

8.26 TELTrackerBugZillaSender

Implements API access to BugZilla bug tracker

Unit

ESendAPIBugZilla 121

Syntax

```
Code (Delphi)
TELTrackerBugZillaSender = class(TELTrackerJSONSender);
```

Remarks

BugZilla uses XML-RPC API.

Requirements

Bug tracker	BugZilla 3 or above

8.27 TELTrackerExceptionlessSender

Implements API access to Exceptionless bug tracker

Unit

ESendAPIExceptionless 122

Syntax

```
Code (Delphi)
TELTrackerExceptionlessSender = class(TELTrackerJSONSender);
```

Remarks

Exceptionless uses REST JSON API.

Requirements

Bug tracker	Exceptionless 4 or above

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.28 TELTrackerFogBugzSender

Implements API access to Fogbugz bug tracker

Unit

ESendAPIFogBugz 122

Syntax

```
Code (Delphi)
TELTrackerFogBugzSender = class(TELTrackerSender);
```

Remarks

Fogbugz uses custom XML API.

Requirements

Bug tracker	Fogbugz 6 or above

8.29 TELTrackerFogBugzSender

Enter topic text here.

8.30 TELTrackerGitHubSender

Implements API access to GitHub bug tracker

Unit

ESendAPIGitHub 123

Syntax

```
Code (Delphi)
TELTrackerGitHubSender = class(TELTrackerSender);
```

Remarks

GitHub uses REST JSON API.

Requirements

Bug tracker	GitHub 5 or above

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.31 TELTrackerGitLabSender

Implements API access to GitLab bug tracker

Unit

ESendAPIGitLab 124

Syntax

```
Code (Delphi)
TELTrackerGitLabSender = class(TELTrackerJSONSender);
```

Remarks

GitLab uses REST JSON API.

Requirements

Bug tracker	GitLab 5 or above

8.32 TELTrackerJIRASender

Implements API access to JIRA bug tracker

Unit

ESendAPIJIRA 124

Syntax

```
Code (Delphi)
TELTrackerJIRASender = class(TELTrackerJSONSender);
```

Remarks

JIRA uses REST JSON API.

Requirements

Bug tracker	JIRA 5 or above

8.33 TELTrackerMantisSender

Implements API access to Mantis bug tracker

Unit

ESendAPIMantis 125

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELTrackerMantisSender = class(TELTrackerSender);
```

Remarks

Mantis uses SOAP API.

Requirements

Bug tracker	Mantis 1.2.7 or above

8.34 TELTrackerMantisSender

Enter topic text here.

8.35 TELTrackerRedmineSender

Implements API access to Redmine bug tracker

Unit

ESendAPIRedmine 126

Syntax

```
Code (Delphi)
TELTrackerRedmineSender = class(TELTrackerJSONSender);
```

Remarks

Redmine uses REST JSON API.

Requirements

Bug tracker	Redmine 2.4 or above

8.36 TELUniversalSender

Base class for all send methods.

Unit

ESend 120

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
TELUniversalSender = class(TObject);
```

Remarks

This is base abstract class for send methods in EurekaLog. It provides service methods and implements common interface.

Actual (non abstract) send methods are implemented as child sub-classes for this class. Each send method class must be registered via RegisterSender 520 or RegisterSenderFirst 520 functions.

Each child class must implement at least SendMessage method (public), where it must implement sending functionality.

8.37 TEurekaBaseStackList

Base call stack class.

Unit

ECallStack 59

Syntax

```
TEurekaBaseStackList = class(TCheckedInterfacedPersistent);
```

Remarks

This is parent class for any call stack builder/tracer classes.

Note

This is base abstract class. Do not create instances of this class. Use EurekaCallStackClass 755 to create default call stack class or use GetTracerClass 431 to get specific class.

Methods 173	Methods
public	Public members

See also

- TEurekaStackListV6 class 217
- TEurekaStackListV7 class 218
- TGenericStackList class 249
- EurekaCallStackClass global variable 755
- GetTracer function 430
- GetTracerClass function 431

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.37.1 Methods

GetCurrentInstruction 173	Returns current instraction for current thread.
GetCurrentStackFrame 174	Returns current CPU stack frame for current thread.
GetCurrentStackPointer 175	Returns current CPU stack pointer for current thread.
GetCurrentTopOfStack 175	Returns top of CPU stack for current thread.
Create 176	Creates call stack object instance and (optionally) builds/traces stack.
Build 178	Builds call stack.

See also

- TEurekaBaseStackList 172
- ECallStack 59

8.37.1.1 GetCurrentInstruction

Returns current instraction for current thread.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
class function GetCurrentInstruction: Pointer;
   static;
```

Return value

Pointer to code segment, which points to current execution point.

This will be the pointer to code which called this method (`GetCurrentInstruction`).

Remarks

Use this routine to obtain current execution point.

This method is often used when building call stack. Call stack includes callers (because it's **call** stack), but it do not hold current instruction. So, you can use this method to obtain current execution point and pass it as `AFirstAddr` value.

Examples

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerWindows);
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- GetCurrentTopOfStack method 175
- GetCurrentStackPointer method 175
- GetCurrentStackFrame method 174
- TEurekaBaseStackList Class 172

8.37.1.2 GetCurrentStackFrame

Returns current CPU stack frame for current thread.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
class function GetCurrentStackFrame: ETypes.PStackFrame;
   static;
```

Return value

Pointer to stack's data, which points to current stack frame.

Remarks

This value is very similar to GetCurrentStackPointer 175, except it is adjusted to point to stack's frame block.

x86-64: use `GetCurrentStackPointer` instead.

Tip

Don't use this function to walk through stack. Use Caller or GetTracer 430.

See also

- GetCurrentTopOfStack method 175
- GetCurrentStackFrame method 174
- GetCurrentInstruction method 173
- TEurekaBaseStackList Class 172

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.37.1.3 GetCurrentStackPointer

Returns current CPU stack pointer for current thread.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
class function GetCurrentStackPointer: Pointer;
   static;
```

Return value

Pointer to stack's data, which points to current stack position.

Remarks

Current position is most recent position of the stack. Since stack is growing backwards, this value points to the logical end of the stack, but physical beginning.

See also

- GetCurrentTopOfStack method 175
- GetCurrentStackFrame method 174
- GetCurrentInstruction method 173
- TEurekaBaseStackList Class 172

8.37.1.4 GetCurrentTopOfStack

Returns top of CPU stack for current thread.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
class function GetCurrentTopOfStack: Pointer;
   static;
```

Return value

Pointer to stack's data, which points to top of stack.

Remarks

Top of stack is greatest pointer possible. Since stack is growing backwards, this value points to the logical beginning of the stack, but physical end.

See also

- GetCurrentStackPointer method
- GetCurrentStackFrame method 174
- GetCurrentInstruction method 173

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- TEurekaBaseStackList Class 172

8.37.1.5 Create

Creates call stack object instance and (optionally) builds/traces stack.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
constructor Create(
   const AFirstAddr: Pointer;
   const AStackFrame: Pointer = nil;
   const ATopOfStack: Pointer = nil;
   const AStartLevel: Integer = -1;
   const ALevelsNumber: Integer = -1;
   const ADebugDetails: TEurekaDebugDetails 680 = [ddUnit, ddProcedure, ddSour
   const AThreadID: DWord = 0;
   const ARunningThread: Boolean = True;
   const AGetDebugInfo: Boolean = True;
   const AShowBeginCalls: Boolean = False;
   const AAllowedMethods: DWord = TracerInvalidMask;
   const AStackOffset: PtrUInt 609 = 0;
   const AContext: Pointer = nil
); overload; virtual;

constructor Create;
   overload;
   virtual;
```

Parameters

AFirstAddr [in, optional]

Code address to add to stack. May be `nil`, which means do not add any additional items to stack.

When specified:

- must point to a code location (not to the stack).
- the specified location will be added as first item to final call stack

AStackFrame [in, optional]

Specifies current stack pointer. This value indicate starting pointer to start tracing process. May be `nil` (default), which means use current stack pointer (i.e. trace entire stack).

When specified:

- must point to stack.
- this value must point to a valid stack frame.
- this value must be less than `ATopOfStack`.

ATopOfStack [in, optional]

Specifies top of the stack. This value indicate ending value for tracing process. May be `nil` (default), which means use current stack limits (i.e. trace entire stack).

Usually this value is not specified, unless you want to build call stack for another (external) thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

When specified:
- must point to stack.
- must be greater than `AStackFrame`.

`AStartLevel` [in, optional]
Indicates how many items (i.e. levels) should be skipped from start of call stack. Default is -1, which means "do not remove any items".

`ALevelsNumber` [in, optional]
Limits length (items' count) of call stack. Default is -1, which means "do not limit call stack".

`ADebugDetails` [in, optional]
Only store locations with this level of details. Default is `[ddUnit, ddProcedure, ddSourceCode]`.

`AThreadID` [in, optional]
A thread ID (this is not thread handle).

Mark call stack items as belonging to the specified thread. Default is 0, which means current thread.

This argument is used to build call stack for another (external) thread.

`ARunningThread` [in, optional]
Set this argument to `True` when build call stacks manually. Value of `False` is used to indicate error (exception) thread. Default is `True`.

`AGetDebugInfo` [in, optional]
Retrieve debug information.

`False` - retrieve only RAW pointers; UnitName, ClassName, ProcedureName and LineNumber will be empty.

`True` - retrieve full information. Each possible property will be filled. This works slower.

`AShowBeginCalls` [in, optional]
For backward compatibility only. Always use `False`.

`AAllowedMethods` [in, optional]
Mask for call stack tracing methods to use. Default is DefaultTracerMask 753 (which is "EurekaLog 7 RAW"). Do not use this argument. Use GetTracer 430 or GetTracerClass 431 functions.

Remarks

This is object constructor, which creates instances of call stack.

Warning

Do not create instances of TEurekaBaseStackList class. Use EurekaCallStackClass 755 to create default call stack class or use GetTracerClass 431 to get specific class.

Note

There are two overloaded `Create` constructors. `Create` without arguments will create object instance without building call stack. You can trace stack on request by using Build 178 method. `Create` with arguments will create object instance AND build call stack. This overload is equal to `Create` + `Build`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEurekaBaseStackList Class 172

8.37.1.6 Build

Builds call stack.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
public
procedure Build(
   const AFirstAddr: Pointer = nil;
   const AStackFrame: Pointer = nil;
   const ATopOfStack: Pointer = nil;
   const AStartLevel: Integer = -1;
   const ALevelsNumber: Integer = -1;
   const ADebugDetails: TEurekaDebugDetails 680 = [ddProcedure, ddUnit, ddSour
   const AThreadID: DWord = 0;
   const ARunningThread: Boolean = True;
   const AGetDebugInfo: Boolean = True;
   const AShowBeginCalls: Boolean = False;
   const AMethods: DWord = TracerInvalidMask;
   const AStackOffset: PtrUInt 609 = 0;
   const AContext: Pointer = nil
);
```

Parameters

AFirstAddr [in, optional]

Code address to add to stack. May be `nil` (default), which means do not add any additional items to stack.

When specified:

- must point to a code location (not to the stack).
- the specified location will be added as first item to final call stack

AStackFrame [in, optional]

Specifies current stack pointer. This value indicate starting pointer to start tracing process. May be `nil` (default), which means use current stack pointer (i.e. trace entire stack).

When specified:

- must point to stack.
- this value must point to a valid stack frame.
- this value must be less than `ATopOfStack`.

ATopOfStack [in, optional]

Specifies top of the stack. This value indicate ending value for tracing process. May be `nil` (default), which means use current stack limits (i.e. trace entire stack).

Usually this value is not specified, unless you want to build call stack for another (external) thread.

When specified:

- must point to stack.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- must be greater than `AStackFrame`.

`AStartLevel` [in, optional]
Indicates how many items (i.e. levels) should be skipped from start of call stack. Default is -1, which means "do not remove any items".

`ALevelsNumber` [in, optional]
Limits length (items' count) of call stack. Default is -1, which means "do not limit call stack".

`ADebugDetails` [in, optional]
Only store locations with this level of details. Default is `[ddUnit, ddProcedure, ddSourceCode]`.

`AThreadID` [in, optional]
A thread ID (this is not thread handle).

Mark call stack items as belonging to the specified thread. Default is 0, which means current thread.

This argument is used to build call stack for another (external) thread, assuming that you pass CPU context and other values belonging to the specified thread. Please note that target thread must be a current thread or a suspended thread.

However, if you want to build a call stack for another (external) thread - there is an easier way: use the `BuildForThread` method.

`ARunningThread` [in, optional]
Set this argument to `True` when build call stacks manually. Value of `False` is used to indicate error (exception) thread. Default is `True`.

`AGetDebugInfo` [in, optional]
Retrieve debug information.

`False` - retrieve only RAW pointers; UnitName, ClassName, ProcedureName and LineNumber will be empty.

`True` - retrieve full information. Each possible property will be filled. This works slower.

`AShowBeginCalls` [in, optional]
For backward compatibility only. Always use `False`.

`AMethods` [in, optional]
Mask for call stack tracing methods to use. Default is DefaultTracerMask 753 (which is "EurekaLog 7 RAW"). Do not use this argument. Use GetTracer 430 or GetTracerClass 431 functions.

`AContext` [in, optional]
A pointer to CPU context record. Supply this value when available.

Remarks

Traces CPU stack and looks for return addresses. Each found return address will be added to the list (object). You can walk though stack by using Count and Items properties.

See also

- BuildForThread method
- BuildForThreadByID Method
- TEurekaBaseStackList Class 172

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.38 TEurekaExceptionInfo

Exception information class.

Unit

EException 89

Syntax

Code (Delphi)

```
TEurekaExceptionInfo = class(TInterfacedPersistent, IInterface);
```

Remarks

This is class for storing exception information. It's used everywhere in EurekaLog when you need to work with exceptions.

Methods 180	Methods
Properties 181	Properties
public	Public members

See also

- ExceptionManager function 396

8.38.1 Methods

ToString 180	String representation of exception information.

See also

- TEurekaExceptionInfo 180
- EException 89

8.38.1.1 ToString

String representation of exception information.

Unit

EException 89

Syntax

Code (Delphi)

```
public
function ToString: String;
   override;
```


© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Exception message and compact call stack.

Remarks

Use this property to get simple human-redable description of the exception.

Tip
Use ELogBuilder 101 and/or ESysInfo 134 units to build more advanced logs.

See also

- ExceptionManager 396
- TEurekaExceptionInfo Class 180

8.38.2 Properties

ActiveControl 183	Window handle of active control.
ActiveControlClass 184	Class name of active control.
ActiveControlText 184	Text of active control.
ActiveForm 184	Window handle of active window.
ActiveFormClass 185	Class name of active window.
ActiveFormText 185	Text of active window.
AdditionalInfo 186	Custom data.
Address 186	Exception address.
BaseException 186	Base exception.
BugID 187	BugID value.
BugIDSource 188	Source information for BugID.
BugIDStr 188	String representation of BugID property.
CallStack 188	Exception call stack.
Context 189	CPU state.
Cursor 189	Mouse cursor.
CursorPos 189	Mouse cursor position.
CustomID 190	Custom part of BugID value.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ExceptionClass 190	Exception class.
ExceptionCode 190	Exception error code.
ExceptionExternal 191	Indicates if exception object is created by external module (DLL).
ExceptionLocation 191	Exception address as text.
ExceptionMessage 192	Exception message.
ExceptionNative 192	Indicates if exception is compatible with current RTL.
ExceptionNumber 193	Number of exceptions.
ExceptionObject 193	Exception object.
ExceptionRecord 195	Exception record.
ExceptionThreadID 195	No description available
Expected 195	Indicates type of exception (expected).
ExpectedContext 196	Help context number for exception.
ExpectedURL 197	Exception URL.
Handled 197	Indicates type of exception (handled).
Handler 198	Exception handler name.
Info 198	Returns low-level exception information.
InitializationException 199	Indicates type of exception (initialization).
InnerException 199	Inner exception.
LastSystemErrorCode 200	Value of `GetLastError` function when exception was raised.
Module 200	Exception module.
ModuleName 200	Exception module name.
ModuleVersion 201	Version of exception module.
Options 201	EurekaLog options for this exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Processed 202	Indicates type of exception (processed).
Reader 202	Exception information reader.
ReproduceText 203	Steps to reproduce.
SafeCallExpt 203	Indicates type of exception (safecall).
Secondary 204	Indicates type of exception (secondary).
StackDump 204	Stack dump.
StackPoint 205	Indicates current position of stack.
StackTop 205	Indicates top of stack.
SynchronizeException 206	Indicates type of exception (synchronize).
ThreadException 206	Indicates type of exception (thread).
ThreadID 207	No description available
Time 208	Exception time.
UnhandledExceptionNumber 208	Number of unhandled exceptions.

See also

- TEurekaExceptionInfo 180
- EException 89

8.38.2.1 ActiveControl

Window handle of active control.

Unit

EException 89

Value

Window handle of active control inside active window. Can be 0.

Remarks

Contains information about GUI elements when exception was raised.

Tip
This handle may no longer be valid when you read this property. Control may be deleted right after exception was raised.

See also

- ActiveFormClass property 185

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ActiveFormText property 185
- ActiveControlClass property 184
- ActiveControlText property 184
- ActiveForm property 184
- TEurekaExceptionInfo Class 180

8.38.2.2 ActiveControlClass

Class name of active control.

Unit
EException 89

Value
Window class name. Such as "EDIT".

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormClass property 185
- ActiveFormText property 185
- ActiveControlText property 184
- ActiveControl property 183
- TEurekaExceptionInfo Class 180

8.38.2.3 ActiveControlText

Text of active control.

Unit
EException 89

Value
Arbitrary text from active control.

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormClass property 185
- ActiveFormText property 185
- ActiveControlClass property 184
- ActiveControl property 183
- TEurekaExceptionInfo Class 180

8.38.2.4 ActiveForm

Window handle of active window.

Unit
EException 89

Value
Window handle of active window. Can be 0.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Contains information about GUI elements when exception was raised.

Tip
This handle may no longer be valid when you read this property. Window may be deleted right after exception was raised.

See also

- ActiveFormClass property 185
- ActiveFormText property 185
- ActiveControlClass property 184
- ActiveControlText property 184
- ActiveControl property 183
- TEurekaExceptionInfo Class 180

8.38.2.5 ActiveFormClass

Class name of active window.

Unit

EException 89

Value

Window class name. Such as "TForm1".

Remarks

Contains information about GUI elements when exception was raised.

See also

- ActiveFormText property 185
- ActiveControlClass property 184
- ActiveControlText property 184
- ActiveForm property 184
- TEurekaExceptionInfo Class 180

8.38.2.6 ActiveFormText

Text of active window.

Unit

EException 89

Value

Arbitrary text from active window.

Remarks

Contains information about GUI elements when exception was raised.

See also

- ActiveFormClass property 185
- ActiveControlClass property 184
- ActiveControlText property 184
- ActiveForm property 184
- TEurekaExceptionInfo Class 180

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.38.2.7 AdditionalInfo

Custom data.

Unit
EException 89

Value
Arbitrary user-supplied data for the exception.

Remarks
This value is not used by EurekaLog and can be set to any value for user purposes.

Note
This property can be used to override exception message 192 for dialogs. See UseExceptionComments option.

See also
- TEurekaExceptionInfo Class 180

8.38.2.8 Address

Exception address.

Unit
EException 89

Value
Pointer to code segment. It points to CPU's instruction which raised the exception.

Remarks
This is exception address. It points to exact code location of exception. If you want a human-readable description instead of RAW address - use the .ExceptionLocation 191 property.

This value is added as top item to call stack to indicate exact location of exception. This valus is also listed in bug report on "CPU" page as "EXP" field.

Note
This value is adjusted for software exceptions, so it will point to "call raise" instruction of "raise Exception" Pascal code line.

See also
- ExceptionLocation property 191
- CallStack property 188
- Module property 200
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.9 BaseException

Base exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EException 89

Value
Exception information for base exception. If no such exception exists - returns `nil`.

Remarks
`BaseException` specifies the innermost exception.

Read the value of the `BaseException` property to obtain the innermost exception. If there are no inner exceptions that triggered the current one, `BaseException` returns the current exception; otherwise `BaseException` inspects the inner exceptions until it finds the first one that triggered the chain of exceptions.

See also
- InnerException property 195
- TEurekaExceptionInfo Class 180

8.38.2.10 BugID

BugID value.

Unit
EException 89

Value
A hash that unique identifies exception.

Remarks
This is a full BugID value.

BugID is a 4-byte unsigned integer (LongWord) which is used to identify exception. Two exception with the same BugID value are considered to be the same. BugID is used to establish uniqueness for many EurekaLog features (such as "Do not save duplicate exceptions", etc.). BugID is also used by EurekaLog Viewer tool when importing exceptions from multiple bug report files to eliminate duplicates. Two bug reports are considered to be duplicates of each other if they are bug reports about exceptions with the same BugID.

BugID consists of two parts. Hi-word is defined by EurekaLog. Lo-word is not used by EurekaLog and it's left for your customization.

Hi-word of BugID is a CRC16 value of string (" BugID ident 188") which includes application and module identification, exception class and two lines from call stack. "BugID ident" does not include exception message and exact error location (address). Only human-readable names are taken from call stack (unit name, class name and routine name). Thus, BugID will not change when you rebuild your application.

You can alter BugID (either by appending custom lo-word or fully replace it) in OnCustomBugID event handler.

Tip
It's not recommended to alter this value. Use CustomID 190 to alter user part of BugID value.

Tip
Use BugIDStr 188 to get human-readable string representation of BugID value.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- CustomID property 190
- BugIDStr property 188
- BugIDSource property 188
- TEurekaExceptionInfo Class 180

8.38.2.11 BugIDSource

Source information for BugID.

Unit

EException 89

Value

Application description + module description + exception class + call stack (first few items with debug info). Delimited with #0 characters.

Remarks

This is source string for building BugD value (hi-word, i.e. the EurekaLog-part of it).

See also

- BugID property 187
- TEurekaExceptionInfo Class 180

8.38.2.12 BugIDStr

String representation of BugID property.

Unit

EException 89

Value

Hex-encoded BugID property.

Remarks

Use this property to get human-readable string representation of BugID 187.

See also

- BugID property 187
- TEurekaExceptionInfo Class 180

8.38.2.13 CallStack

Exception call stack.

Unit

EException 89

Value

Call stack for exception.

Remarks

This call stack includes address of exception as first item.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.14 Context

CPU state.

Unit

EException 89

Value

Contains CPU information for exception.

Remarks

This property contains low-level CPU information captured from hardware. This is system data type (non-Delphi). It's specific to hardware (CPU architecture).

Tip
It's not recommended to use this property.

See also

- ExceptionRecord property 195
- Reader property 202
- Address property 186
- CallStack property 188
- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.15 Cursor

Mouse cursor.

Unit

EException 89

Value

Cursor descriptor.

Remarks

Indicates mouse cursor shape when exception was raised.

See also

- CursorPos property 189
- TEurekaExceptionInfo Class 180

8.38.2.16 CursorPos

Mouse cursor position.

Unit

EException 89

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
Absolute desktop coordinates of mouse cursor position.

Remarks
Indicates mouse position when exception was raised.

See also
- Cursor property 189
- TEurekaExceptionInfo Class 180

8.38.2.17 CustomID

Custom part of BugID value.

Unit
EException 89

Value
Low-word of BugID property.

Remarks
This property allows you to modify "custom" part of BugID value.

Tip
Changes in this property will result in changes inside BugID property.

See also
- BugID property 187
- TEurekaExceptionInfo Class 180

8.38.2.18 ExceptionClass

Exception class.

Unit
EException 89

Value
Class of exception object (if present). Examples: 'Exception', 'EAccessViolation', 'EStreamError', etc.

Remarks
Class name of original exception object.

See also
- ExceptionMessage property 192
- Address property 186
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.19 ExceptionCode

Exception error code.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EException 89

Value
Error code that represents exception object.

Remarks
This property contains low-level error code associated with this exception. For example, it could be $C0000005 for access violation exception, or $0EEDFADE for Delphi exceptions.

Tip
You can use this property to return from thread function, or pass to functions like `TerminateThread`, `TerminateProcess`. Just don't forget to make a type conversion (avoid range-check errors) if your code requires `Integer` type.

See also
- ExceptionRecord property 195
- TEurekaExceptionInfo Class 180

8.38.2.20 ExceptionExternal

Indicates if exception object is created by external module (DLL).

Unit
EException 89

Value
`True` if you exception object is created in external module (DLL), `False` otherwise.

Remarks
This property indicates if exception object from `.ExceptionObject` was created in another executable module (DLL). Exception may be raised in another module, but exception object can still be created by current module. If you need to check if exception itself was raised in external module - check `.Address` property.

See also
- ExceptionObject property 193
- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.21 ExceptionLocation

Exception address as text.

Unit
EException 89

Value
Textual human-readable description of the exception address. It describes the CPU's instruction which raised the exception.

Remarks
This is exception address. It points to exact code location of exception. If you want a RAW address instead of a

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

human-readable description - use the .Address 186 property.

This value is added as top item to call stack to indicate exact location of exception.

Note
This value is adjusted for software exceptions, so it will point to "call raise" instruction of "raise Exception" Pascal code line.

See also

- Address property 186
- CallStack property 188
- Module property 200
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.22 ExceptionMessage

Exception message.

Unit

EException 89

Value

Error message (if present). Examples: 'Error Message', 'Access violation at address 0040532 in module "Project1.exe". Write at address 00000000', 'Stream read error', etc.

Remarks

Human-readable error message from exception object.

Note
You can alter this property, but doing so will totally overwrite exception message (both for error dialogs and bug reports). If you want to preserve original exception message, but display custom exception message to user - use AdditionalInfo 186 property.

See also

- AdditionalInfo property 186
- ExceptionClass property 190
- Address property 186
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.23 ExceptionNative

Indicates if exception is compatible with current RTL.

Unit

EException 89

Value

`True` if you can safely cast `.ExceptionObject` property to `TObject class, False` otherwise.

Remarks

This property indicates if exception object from `.ExceptionObject` is compatible with RTL in current

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

executable module. It is typically `True`, but may be `False` when exception comes from another module (DLL) compiled with different IDE version. For example, a Delphi 2007 exception being catched in Delphi XE - `ExceptionNative` would be `False`.

Warning

The ExceptionObject 193 property may be `NIL` when `ExceptionNative` is `True`. For example, if you are looking at already deleted inner (nested) exception.

See also

- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.24 ExceptionNumber

Number of exceptions.

Unit

EException 89

Value

Global exception counter for this exception.

Remarks

EurekaLog keeps global exception counter. Each time an exception occurs - this counter is increased and its current value is saved to this property.

This property is updated when exception is raised. It doesn't matter if exception will be handled, processed, etc.

See also

- UnhandledExceptionNumber 208
- TExceptionManager.ExceptionCount 244
- TEurekaExceptionInfo Class 180

8.38.2.25 ExceptionObject

Exception object.

Unit

EException 89

Value

Exception object type-casted as pointer. I.e. it's a pointer to data of exception object.

Remarks

This is pointer to original exception object.

Warning

Since exception may come from another (external) executable module (rare cases, but could happen), then exception object may be not a Delphi object. Do not try to cast this value to Exception or TObject unless you're 100% sure about this.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Use Address 186, ExceptionClass 190 and ExceptionMessage 192 properties instead.

Examples

Code (Delphi)

```
// This will add custom info to bug report
procedure AddMyData(const ACustom: Pointer;
          AExceptionInfo: TEurekaExceptionInfo;
          ALogBuilder: TBaseLogBuilder;
          ADataFields: TStrings;
          var ACallNextHandler: Boolean);

   // Extract native Delphi exception from exception info
   function GetExceptionObject(AExceptionInfo: TEurekaExceptionInfo): Ex
   var
      EObj: TObject;
   begin
      // Exception was thrown by a compatible Delphi version?
      if (AExceptionInfo.ExceptionObject <> nil) and
          AExceptionInfo.ExceptionNative then
         EObj := TObject(AExceptionInfo.ExceptionObject)
      else
         EObj := nil; // rare

      // Exception is SysUtils.Exception?
      if EObj is Exception then
         Result := Exception(EObj)
      else
         Result := nil; // very rare
   end;

var
   E: Exception;
begin
   E := GetExceptionObject(AExceptionInfo);
    // E is either nil (for non-Delphi exceptions or deleted inner excep
    // or E is current Exception

   // Add custom info for custom exceptions
   if E is EMyException then
      ADataFields.Values['Code'] := EMyException(E).Code;
   end;

initialization
   RegisterEventCustomDataRequest(nil, AddMyData);
end.
```

See also

- ExceptionClass property 190
- ExceptionMessage property 192
- ExceptionNative property 192
- ExceptionExternal property 191
- TEurekaExceptionInfo Class 180

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.38.2.26 ExceptionRecord

Exception record.

Unit

EException 89

Value

Contains system information for exception.

Remarks

This property contains low-level system information captured from system. This is system data type (non-Delphi). It's specific to operating system.

Tip
It's not recommended to use this property. Use other properties of `TEurekaExceptionInfo` class, which exposes exception information extracted from this property.

See also

- Context property 189
- Reader property 202
- Address property 186
- CallStack property 188
- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- ExceptionCode property 190
- TEurekaExceptionInfo Class 180

8.38.2.27 ExceptionThreadID

Exception Thread ID (original exception's thread).

Unit

EException 89

Value

ID of thread. It identificates a thread that has originally raised the exception.

Remarks

This value indicates exception thread. This value never changes once set.

See also

- ThreadID property 207
- ThreadException property 206
- Address property 186
- CallStack property 188
- TEurekaExceptionInfo Class 180

8.38.2.28 Expected

Indicates type of exception (expected).

Unit

EException 89

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value

`True` - this is expected exception.

`False` - this is normal/typical exception.

Remarks

This property indicates type of exception - expected or not.

Exceptions that do not represent a bug in your code are called "expected exceptions". You can mark any exception as expected via various methods. Please see this article for examples of each method.

You can set the .ExpectedContext 196 or .ExpectedURL 196 properties to set the `Expected` property to `True`.

When EurekaLog processes expected exception: it disables creation of bug report, send and restart features, as well as showing technical details (call stack, etc.). Basically, processing of expected exception will only show the exception message to the user and nothing else.

See also

- Handled property 197
- Secondary property 204
- Processed property 202
- SafeCallExpt property
- SynchronizeException property 206
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.29 ExpectedContext

Help context number for exception.

Unit

EException 89

Value

Integer for HelpContext. Must be a valid ID of help topic inside help file for your application.

Remarks

Contains help context number. Initially empty. You can set this number to point an existing topic of your help file.

When this property is set, exception is considered to be "known" (expected). The help button will be displayed in dialogs, which will open help file of your application with specified topic ID.

Expected exceptions are handled by EurekaLog, but behave differently. They can not be send to developers and do not generate bug reports, there will be no restart/terminate options, etc. In other words, expected exception are handled as just "error dialogs" with no diagnostic/recovery.

You can set exception as expected (as well as set this property) by using RaiseExpected 499 routine.

Tip
This property is used by "Help" button in dialogs. Setting this property will force "Help" button to appear.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Important!
Altering this property will alter way of handling this exception by EurekaLog.

See also

- ExpectedURL property 197
- Expected property 195
- TEurekaExceptionInfo Class 180

8.38.2.30 ExpectedURL

Exception URL.

Unit

EException 89

Value

Arbitrary URL to any resource (resource is expected to describe this exception).

Remarks

Contains help context number. Initially empty. You can set this number to point an existing topic of your help file.

When this property is set, exception is considered to be "known" (expected).

Expected exceptions are handled by EurekaLog, but behave differently. They can not be send to developers and do not generate bug reports.

You can set exception as expected (as well as set this property) by using RaiseExpected 499 routine.

Tip
This property is displayed by console dialog.

Important!
Altering this property will alter way of handling this exception by EurekaLog.

See also

- ExpectedURL property 197
- Expected property 195
- TEurekaExceptionInfo Class 180

8.38.2.31 Handled

Indicates type of exception (handled).

Unit

EException 89

Value

`True` - this is handled exception.

`False` - this is unhandled exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This property indicates type of exception - handled or not. Handled = `False` on creation and may become `True` later, when handling code will run.

Unhandled exception is exception which wasn't catched by your code. Such exceptions are processed by EurekaLog.

Handled exception is exception which was catched by your application. Such exceptions are processed by your code and EurekaLog is not invoked for them (by default). You can alter this behavior by using "catch handled exception" option in EurekaLog project options.

See also

- Processed property 202
- Secondary property 204
- Expected property 195
- SafeCallExpt property
- SynchronizeException property 206
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.32 Handler

Exception handler name.

Unit

EException 89

Value

One of `atXYZ` constants (such as `atVCL`, `atThread`, `atDataSnap`, etc.).

Remarks

This property indicates which handler is processing exception. For example, you can use this property in your event handlers to change dialog from MS Classic to none - if exception was catched by `atThread` or `atDataSnap` (as opposed to atVCL).

Default value is `atGeneric`.

See also

- TEurekaExceptionInfo Class 180

8.38.2.33 Info

Returns low-level exception information.

Unit

EException 89

Value

Low-level exception information.

Remarks

Low-level information that was captured for this exception.

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Do not use this property. Other properties of `TEurekaExceptionInfo` class exposes exception information extracted from this property. Use other properties of `TEurekaExceptionInfo` class to get exception information.

See also

- ExceptionRecord property 195
- Context property 189
- Reader property 202
- TEurekaExceptionInfo Class 180

8.38.2.34 InitializationException

Indicates type of exception (initialization).

Unit

EException 89

Value

`True` - this is initialization/finalization exception.

`False` - this is normal exception.

Remarks

This property indicates type of exception - initialization/finalization or not.

Initialization/finalization exception is an exception inside initialization/finalization code blocks of unit. Such exceptions are always unhandled.

See also

- Processed property 202
- Handled property 197
- Secondary property 204
- Expected property 195
- SafeCallExpt property 203
- SynchronizeException property 206
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.35 InnerException

Inner exception.

Unit

EException 89

Value

Exception information for inner exception. If no such exception exists - returns `nil`.

Remarks

Read the value of the `InnerException` property to obtain the inner exception. If there are no exceptions that triggered the current one, `InnerException` returns `nil`; otherwise `InnerException` returns the inner exception that triggered a specific exception.

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unlike RTL's InnerException, EurekaLog's `InnerException` is always captured.

See also

- BaseException property 186
- TEurekaExceptionInfo Class 180

8.38.2.36 LastSystemErrorCode

Value of `GetLastError` function when exception was raised.

Unit

EException 89

Value

System error code returned by the `GetLastError` function.

Remarks

EurekaLog saves value of the `GetLastError` function into this property when exception was thrown.

See also

- TEurekaExceptionInfo Class 180

8.38.2.37 Module

Exception module.

Unit

EException 89

Value

HINST/HMODULE of executable module containing exception address 186. Can be 0 if exception was raised in dynamically generated code.

Remarks

Handle of the exception module (if present).

Note
Exception module may be unloaded after exception raising, so this value will hold old handle which is no longer valid. Use ModuleName 200 / ModuleVersion 201 properties to get module information.

See also

- ModuleName property 200
- ModuleVersion property 201
- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.38 ModuleName

Exception module name.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EException 89

Value
Full file name of executable module containing exception address 186. Can be '' (empty string) if exception was raised in dynamically generated code.

Remarks
File name of the exception module (if present).

Since exception module may be unloaded after exception raising, this value holds cached module name, so it can be used at any time.

See also
- Module property 200
- ModuleVersion property 201
- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.39 ModuleVersion

Version of exception module.

Unit
EException 89

Value
Version information of executable module containing exception address 186. Can be '' (empty string) if exception was raised in dynamically generated code.

Remarks
Version information of the exception module (if present).

Since exception module may be unloaded after exception raising, this value holds cached version info, so it can be used at any time.

See also
- Module property 200
- ModuleName property 200
- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.40 Options

EurekaLog options for this exception.

Unit
EException 89

Value
Actual EurekaLog options for this exception.

Remarks
Each exception gets its own TEurekaExceptionInfo object which describes this exception. Each TEurekaExceptionInfo object has Options property of type TEurekaModuleOptions. Therefore, each exception

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

has its own EurekaLog options.

Options are isolated and do not depend on CurrentEurekaLogOptions and options of other exceptions. I.e. any change to Options property will be visible only for that exception and nowhere else.

Initially options equal to CurrentEurekaLogOptions, but they may be changed during exception processing.

Other objects (like dialog, sender, log builder, etc.) share this Options property. I.e. all processing code uses the same object instance, so changes in, say, event handler will be visible for all other objects (dialogs, log builder, etc.).

See also

- CurrentEurekaLogOptions
- TEurekaExceptionInfo Class 180

8.38.2.41 Processed

Indicates type of exception (processed).

Unit

EException 89

Value

`True` - this is processed exception.

`False` - this is unprocessed exception.

Remarks

This property indicates type of exception - processed or not.

Processed exception is an exception that was processed (i.e. handled) by EurekaLog. Each exception is initially unprocessed. It becomes processed when EurekaLog handles it via ExceptionManager 396, Handle 234 and ProcessException. Once exception is processed, it's no longer invokes EurekaLog and can not be handled again. Setting `Processed` to `True` will exclude exception from EurekaLog's processing.

See also

- Handled property 197
- Secondary property 204
- Expected property 195
- SafeCallExpt property 203
- SynchronizeException property 206
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.42 Reader

Exception information reader.

Unit

EException 89

Value

Exception reader object used to extract information from exception low-level records.

Remarks

This is instance of so-called "exception information reader" class (which is a child class for

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

`TExceptionReader`). This object was used to fill properties of `TEurekaExceptionInfo` instance.

EurekaLog has several different readers which are able to extract information from Delphi exceptions, C++ Builder exceptions, hardware exceptions. It's possible to add more extractor classes (for example, to support Visual C++ exceptions).

Tip
It's not recommended to use this property. Use other properties of the TEurekaExceptionInfo class.

See also

- ExceptionRecord property 195
- Context property 189
- Address property 186
- CallStack property 188
- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.43 ReproduceText

Steps to reproduce.

Unit

EException 89

Value

Text for steps to reproduce as entered by user.

Remarks

This value is initially empty and it's set by MS Classic or EurekaLog dialogs when user enters non-empty "steps to reproduce".

See also

- TEurekaExceptionInfo Class 180

8.38.2.44 SafeCallExpt

Indicates type of exception (safecall).

Unit

EException 89

Value

`True` - this is safecall exception.

`False` - this is normal exception.

Remarks

This property indicates type of exception - safecall or not.

Safecall exception is an exception inside safecall method. Such exceptions are handled by SafeCallException method of called object (that's why safecall exceptions are always handled).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Processed property 202
- Handled property 197
- Secondary property 204
- Expected property 195
- SynchronizeException property 206
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.45 Secondary

Indicates type of exception (secondary).

Unit

EException 89

Value

`True` - this is secondary exception.

`False` - this is normal/typical/first exception.

Remarks

This property indicates type of exception - secondary or not.

Secondary exception is an exception which was raised (thrown) when another exception is being handled. For example, your code raises exception, EurekaLog shows dialog about your exception. Then your code throws another exception, while dialog is still being displayed. That second exception will be secondary exception.

EurekaLog can be configured to behave differently for secondary exceptions.

See also

- Handled property 197
- Processed property 202
- Expected property 195
- SafeCallExpt property
- SynchronizeException property 206
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.46 StackDump

Stack dump.

Unit

EException 89

Value

Pointer to stack dump.

Remarks

Stack dump is a block of RAW memory. It's copied from CPU stack from StackPoint 205 position. It's size equals SizeOf(Pointer) X EStackDumpDepth (EConsts unit). I.e. 4 x 16 = 64 bytes for x86-32, or 8 x 16 = 128 bytes for x86-64. No padding is used.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- StackPoint property 205
- CallStack property 188
- Address property 186
- TEurekaExceptionInfo Class 180

8.38.2.47 StackPoint

Indicates current position of stack.

Unit

EException 89

Value

Pointer to stack's current position of exception thread.

Remarks

This property contains pointer to current position of CPU stack for the exception thread. It is usually ESP/EBP register.

This value is used to capture dump of CPU stack. It's also listed in bug report on CPU page as "STK" value.

Tip
It's not recommended to use this property.

See also

- StackTop property 205
- ExceptionRecord property 195
- Context property 189
- Address property 186
- CallStack property 188
- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.48 StackTop

Indicates top of stack.

Unit

EException 89

Value

Pointer to top of stack of exception thread.

Remarks

This property contains pointer to top of CPU stack for the exception thread. "Top of the stack" is stack's highest available address.

Tip
It's not recommended to use this property.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- StackPoint property 205
- ExceptionRecord property 195
- Context property 189
- Address property 186
- CallStack property 188
- ExceptionObject property 193
- ExceptionClass property 190
- ExceptionMessage property 192
- TEurekaExceptionInfo Class 180

8.38.2.49 SynchronizeException

Indicates type of exception (synchronize).

Unit

EException 89

Value

`True` - this is synchronize exception.

`False` - this is normal exception.

Remarks

This property indicates type of exception - synchronized or not.

Synchronized exception is an exception inside Synchronize method. Such exceptions are handled by `Synchronize` routine itself (that's why synchronized exceptions are always handled). Synchronized exceptions are re-raised in another thread.

See also

- ThreadID property 207
- ExceptionThreadID property 195
- Processed property 202
- Handled property 197
- Secondary property 204
- Expected property 195
- SafeCallExpt property 203
- InitializationException property 199
- ThreadException property 206
- TEurekaExceptionInfo Class 180

8.38.2.50 ThreadException

Indicates type of exception (thread).

Unit

EException 89

Value

`True` - this exception was raised inside a background thread and it is captured to be re-raised later (in the main thread).

`False` - this is normal exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This property indicates type of exception - thread exception or not.

Thread exception is exception inside thread which is captured by thread wrapper (such as TThread or TOmniTask). Such exceptions are handled by the wrapper by saving exception into "FatalException" property. Such exceptions are not handled inside thread.

This property is always True when SynchronizeException property is True.

See also

- ThreadID property 207
- ExceptionThreadID property 195
- Processed property 202
- Handled property 197
- Secondary property 204
- Expected property 195
- SafeCallExpt property 203
- InitializationException property 199
- SynchronizeException property 206
- TEurekaExceptionInfo Class 180

8.38.2.51 ThreadID

Thread ID (actual exception's thread).

Unit

EException 89

Value

ID of thread. It identificates a thread that has raised or re-raised the exception.

Remarks

This value indicates exception thread. The same code may be run in multiple threads. This property can be used to distinguish between same exceptions in different threads.

Warning

This value identificates a thread that has raised or re-raised the exception. In other words, if you take exception from a background thread (`TThread`) and re-raise it in the main thread - this property will return ID of the main thread, since the main thread has re-raised the exception. If you want to know which thread has originally raised the exception - use the .ExceptionThreadID 195 property instead.

Warning

`.ThreadID` does not always match `GetCurrentThreadID` function. For example, a stack overflow exception will have `.ThreadID` set to the thread which has raised the stack overflow exception. However, the `GetCurrentThreadID` function will return a different ID, because all stack overflow exceptions are being handled in a standalone service background thread.

Tip

In most cases, but **not** always: `.ThreadID = .ExceptionThreadID = GetCurrentThreadID`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ExceptionThreadID property 195
- ThreadException property 206
- Address property 186
- CallStack property 188
- TEurekaExceptionInfo Class 180

8.38.2.52 Time

Exception time.

Unit

EException 89

Value

Date-time of exception raising. Local time.

Remarks

Indicates when exception was raised.

See also

- TEurekaExceptionInfo Class 180

8.38.2.53 UnhandledExceptionNumber

Number of unhandled exceptions.

Unit

EException 89

Value

Global unhandled exception counter for this exception.

Remarks

EurekaLog keeps global exception counter. Each time an exception is handled by EurekaLog - this counter is increased and its current value is saved to this property (and in bug report as "Exception/Count" field).

This property is updated when exception is handled by EurekaLog. It's not set until that moment.

See also

- ExceptionNumber 193
- TExceptionManager.UnhandledExceptionCount 245
- TEurekaExceptionInfo Class 180

8.39 TEurekaLogDetailedDialog

Enter topic text here.

8.40 TEurekaLogDialog

Enter topic text here.

8.41 TEurekaLogEvents

EurekaLog 7 component.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EComponent 62

Syntax

Code (Delphi)

```
TEurekaLogEvents = class(TComponent);
```

Remarks

EurekaLog 7 component can be used as simple way to react to EurekaLog's events.

This component provides a quick access to EurekaLog events. Normally a EurekaLog event handler is procedure or method which are registered (subscribed) to events via routines in EEvents 83 unit. By using this component you can just double-click on events in IDE's Object Inspector to create handlers. No code writing is necessary.

Caution!

While using component is very easy, you should be aware that any component lives as long as its container (form, frame or data module) lives. Thus, any event handler which you create with component will be inactive after your form close.

For this reason it's best to avoid component at all and use event handlers directly. Any event handler can be registered via EEvents 83 unit. Best place to register event handler - `initialization` sections of your units.

If you still want to use EurekaLog component - best option is to place it on the main form of your application.

Note

It's perfectly fine to have multiple EurekaLog components in your application.

Events 34	Events
private	Private members
published	Published members

Examples

For sample code of events handlers - please see description of specific events.

See also

- Events index 11

8.41.1 Events

OnAttachedFilesRequest 210	Wrapper for OnAttachedFilesRequest 21

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	event.
OnBeginReportGen 211	Wrapper for OnBeginReportGen 30 event.
OnCustomBugID 211	Wrapper for OnCustomBugID 25 event.
OnCustomButtonClick 212	Wrapper for OnCustomButtonClick 28 event.
OnCustomDataRequest 212	Wrapper for OnCustomDataRequest 26 event.
OnCustomFileName 212	Wrapper for OnCustomFileName 24 event.
OnCustomWebFieldsRequest 213	Wrapper for OnCustomWebFieldsRequest 27 event.
OnEndReportGen 213	Wrapper for OnEndReportGen 30 event.
OnExceptionAction 214	Wrapper for OnExceptionAction 16 event.
OnExceptionError 214	Wrapper for OnExceptionError 18 event.
OnExceptionNotify 214	Wrapper for OnExceptionNotify 14 event.
OnPasswordRequest 215	Wrapper for OnPasswordRequest 19 event.
OnRaise 215	Wrapper for OnRaise 12 event.
OnReproduceRequest 215	Wrapper for OnReproduceRequest 23 event.
OnZippedFilesRequest 216	Wrapper for OnZippedFilesRequest 22 event.

See also

- TEurekaLogEvents 33
- EComponent 62

8.41.1.1 OnAttachedFilesRequest

Wrapper for OnAttachedFilesRequest 21 event.

Unit

EComponent 62

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
User-supplied event handler.

Remarks
For the detailed description please see OnAttachedFilesRequest event 21.

Examples
For code example please see OnAttachedFilesRequest event 21.

See also
- Events 11
- OnAttachedFilesRequest event 21
- TEurekaLogEvents Class 33

8.41.1.2 OnBeginReportGen

Wrapper for OnBeginReportGen 30 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnBeginReportGen event 30.

Examples
For code example please see OnBeginReportGen event 30.

See also
- Events 11
- OnBeginReportGen event 30
- TEurekaLogEvents Class 33

8.41.1.3 OnCustomBugID

Wrapper for OnCustomBugID 25 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnCustomBugID event 25.

Examples
For code example please see OnCustomBugID event 25.

See also
- Events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- OnCustomBugID event 25
- TEurekaLogEvents Class 33

8.41.1.4 OnCustomButtonClick

Wrapper for OnCustomButtonClick 28 event.

Unit

EComponent 62

Value

User-supplied event handler.

Remarks

For the detailed description please see OnCustomButtonClick event 28.

Examples

For code example please see OnCustomButtonClick event 28.

See also

- Events 11
- OnCustomButtonClick event 28
- TEurekaLogEvents Class 33

8.41.1.5 OnCustomDataRequest

Wrapper for OnCustomDataRequest 26 event.

Unit

EComponent 62

Value

User-supplied event handler.

Remarks

For the detailed description please see OnCustomDataRequest event 26.

Examples

For code example please see OnCustomDataRequest event 26.

See also

- Events 11
- OnCustomDataRequest event 26
- TEurekaLogEvents Class 33

8.41.1.6 OnCustomFileName

Wrapper for OnCustomFileName 24 event.

Unit

EComponent 62

Value

User-supplied event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks
For the detailed description please see OnCustomFileName event 24.

Examples
For code example please see OnCustomFileName event 24.

See also
- Events 11
- OnCustomFileName event 24
- TEurekaLogEvents Class 33

8.41.1.7 OnCustomWebFieldsRequest

Wrapper for OnCustomWebFieldsRequest 27 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnCustomWebFieldsRequest event 27.

Examples
For code example please see OnCustomWebFieldsRequest event 27.

See also
- Events 11
- OnCustomWebFieldsRequest event 27
- TEurekaLogEvents Class 33

8.41.1.8 OnEndReportGen

Wrapper for OnEndReportGen 30 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnEndReportGen event 30.

Examples
For code example please see OnEndReportGen event 30.

See also
- Events 11
- OnEndReportGen event 30
- TEurekaLogEvents Class 33

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.41.1.9 OnExceptionAction

Wrapper for OnExceptionAction 16 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnExceptionAction event 16.

Examples
For code example please see OnExceptionAction event 16.

See also
- Events 11
- OnExceptionAction event 16
- TEurekaLogEvents Class 33

8.41.1.10 OnExceptionError

Wrapper for OnExceptionError 18 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnExceptionError event 18.

Examples
For code example please see OnExceptionError event 18.

See also
- Events 11
- OnExceptionError event 18
- TEurekaLogEvents Class 33

8.41.1.11 OnExceptionNotify

Wrapper for OnExceptionNotify 14 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnExceptionNotify event 14.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

For code example please see OnExceptionNotify event 14.

See also

- Events 11
- OnExceptionNotify event 14
- TEurekaLogEvents Class 33

8.41.1.12 OnPasswordRequest

Wrapper for OnPasswordRequest 19 event.

Unit

EComponent 62

Value

User-supplied event handler.

Remarks

For the detailed description please see OnPasswordRequest event 19.

Examples

For code example please see OnPasswordRequest event 19.

See also

- Events 11
- OnPasswordRequest event 19
- TEurekaLogEvents Class 33

8.41.1.13 OnRaise

Wrapper for OnRaise 12 event.

Unit

EComponent 62

Value

User-supplied event handler.

Remarks

For the detailed description please see OnRaise event 12.

Examples

For code example please see OnRaise event 12.

See also

- Events 11
- OnRaise event 12
- TEurekaLogEvents Class 33

8.41.1.14 OnReproduceRequest

Wrapper for OnReproduceRequest 23 event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnReproduceRequest event 23.

Examples
For code example please see OnReproduceRequest event 23.

See also
- Events 11
- OnReproduceRequest event 23
- TEurekaLogEvents Class 33

8.41.1.15 OnZippedFilesRequest

Wrapper for OnZippedFilesRequest 22 event.

Unit
EComponent 62

Value
User-supplied event handler.

Remarks
For the detailed description please see OnZippedFilesRequest event 22.

Examples
For code example please see OnZippedFilesRequest event 22.

See also
- Events 11
- OnZippedFilesRequest event 22
- TEurekaLogEvents Class 33

8.42 TEurekaModuleOptions

EurekaLog options.

Unit
EClasses 61

Syntax

Code (Delphi)

```
TEurekaModuleOptions = class(TInterfacedPersistent);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This class represents EurekaLog options. EurekaLog options are usually stored as one instance per one loaded executable module. However, you can also create a new instance of this class on your request (to create custom options).

Usually you work with this class via CurrentEurekaLogOptions 8 global function or via `Options` property of other EurekaLog classes (such as dialogs, call stacks, etc.).

Options are stored as named pairs text (name=value). Each option occupies one line only. Value is encoded to remove special characters (non-printable, line breaks, etc.). Complex values can be cached.

Note
If you need to constantly use some option value - consider to save it into variable to optimize performance.

For a list of properties - please, refer to description of EurekaLog project options UI. Description of almost every option contains name of corresponding property of `TEurekaModuleOptions` class. For example:

Note
1. "**Save bug report to file**" (`.SaveLogFile`) option enables saving...

This means that you can change this option at run-time by altering `.SaveLogFile` property of either global CurrentEurekaLogOptions 8 or local `Options` property of exception object in event handlers.

8.43 TEurekaStackListV6

Stack tracing class for EurekaLog 6 method.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
TEurekaStackListV6 = class(TEurekaBaseStackList 172);
```

Remarks

Use this class to build call stacks by using EurekaLog 6 method.

Tip
Avoid using classes directly. Use EurekaCallStackClass 755 / GetTracer 430 / GetTracerClass 431

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
begin
   CallStack := GetTracer(TracerRawEurekaLogV6); // preferable
    // Or:
    // CallStack := TEurekaStackListV6.Create; // not recommended
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TEurekaStackListV7 class 218
- TGenericStackList class 249

8.44 TEurekaStackListV7

Stack tracing class for EurekaLog 7 RAW method.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
TEurekaStackListV7 = class(TEurekaBaseStackList 172);
```

Remarks

Use this class to build call stacks by using EurekaLog 7 RAW method.

Tip

Avoid using classes directly. Use EurekaCallStackClass 755/ GetTracer 430/GetTracerClass 431.

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerRawEurekaLogV7); // preferable
    // Or:
    // CallStack := TEurekaStackListV7.Create; // not recommended
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TEurekaStackListV6 class 217
- TGenericStackList class 249

8.45 TEurekaThreadData

Thread information.

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
TEurekaThreadData = class(TCheckedInterfacedObject);
```

Remarks

This class represents information about running thread in your application. Instances of this class can be acquired via ThreadData 577, AcquireThreadData 318 or ThreadDataEnum 578 functions.

Methods 219	Methods
Properties 220	Properties
public	Public members

See also

- ThreadData Function 577
- AcquireThreadData Function 318
- ThreadDataEnum Function 578
- TEurekaThreadDataList Class 229

8.45.1 Methods

Priority 220	No description available

See also

- TEurekaThreadData 219
- EThreadsManager 136

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.45.1.1 Priority

Thread's priority.

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
public
function Priority: Integer;
```

Return value

Value returned by GetThreadPriority function. Possible range is -15..15.

Remarks

This method retrieves the priority value for the thread. This value, together with the priority class of the thread's process, determines the thread's base-priority level.

The thread's priority level can be any number within -15..+15 range. A well-known standard values are:

- -15: (THREAD_PRIORITY_IDLE) Base priority of 1 for usual processes or 16 for REALTIME_PRIORITY_CLASS processes.
- -1: (THREAD_PRIORITY_BELOW_NORMAL) Priority 1 point below the priority class.
- -2: (THREAD_PRIORITY_LOWEST) Priority 2 points below the priority class.
- 0: (THREAD_PRIORITY_NORMAL) Normal priority for the priority class.
- 1: (THREAD_PRIORITY_ABOVE_NORMAL) Priority 1 point above the priority class.
- 2: (THREAD_PRIORITY_HIGHEST) Priority 2 points above the priority class.
- 15: (THREAD_PRIORITY_TIME_CRITICAL) Base-priority level of 15 for usual processes or 31 for REALTIME_PRIORITY_CLASS processes.

See also

- TEurekaThreadData Class 219

8.45.2 Properties

CallerAddress 221	Thread creator (address).
CallerFunction 222	Thread creator (description).
Handle 222	Thread handle.
ID 223	TID.
IsMainThread 223	Indicate whenever this thread is main thread.
IsRTLThread 223	Indicate whenever this thread is RTL thread.
IsWindowsThread 224	Indicate whenever this thread is Windows thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Kind 224	Thread kind (Windows/RTL thread).
Name 225	Thread name (description).
Parameter 225	No description available
ParentID 226	Parent thread ID.
RealParameter 226	No description available
RealStartingAddress 226	Real thread function (pointer).
RealStartingFunction 227	Real thread function (description).
StackTop 227	No description available
StartingAddress 228	Usual thread function (pointer).
StartingFunction 228	Usual thread function (description).
Thread 229	Thread object (class instance).
ThreadClassName 229	Thread class name.

See also

- TEurekaThreadData 219
- EThreadsManager 136

8.45.2.1 CallerAddress

Thread creator (address).

Unit

EThreadsManager 136

Value

A pointer to code which have called CreateThread/BeginThread/TThread.Create. This pointer points somewhere inside a function. It does not point to a function.

Remarks

This property is used to identify thread's creator. This value is used for building multi-threaded call stacks: it is inserted as a very first (bottom) entry in thread's call stack to indicate thread's creator.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- RealStartingAddress Property 226
- CallerFunction Property 222
- Thread Property 229
- ThreadClassName Property 229
- TEurekaThreadData Class 219

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.45.2.2 CallerFunction

Thread creator (description).

Unit

EThreadsManager 136

Value

A human-readable description of CallerAddress 221 property.

Remarks

This value contains function name identified in CallerAddress 221 property. Function name is taken from debug information.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- RealStartingFunction Property 227
- CallerAddress Property 221
- Thread Property 229
- ThreadClassName Property 229
- TEurekaThreadData Class 219

8.45.2.3 Handle

Thread handle.

Unit

EThreadsManager 136

Value

A thread handle with THREAD_QUERY_INFORMATION or THREAD_TERMINATE or SYNCHRONIZE rights or better (usually: full access).

Remarks

This is opened handle for the thread. The handle is opened on thread's start and is closed on thread's termination. It is not changed during thread's life-time.

Note
This handle is not equal to a return value for CreateThread/BeginThread functions or TThread.Handle property.

Warning
Handle could not be used to identify threads. A single thread may have several different handles opened. Use ID 223 property to identify threads.

See also

- ID Property 223
- ParentID Property 226
- Name Property 225
- TEurekaThreadData Class 219

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.45.2.4 ID

TID.

Unit
EThreadsManager 136

Value
Thread ID.

Remarks
This property stores TID which unique identifies thread in a running application. TID is used as a key for thread lists.

TID is detected on thread's creation. It is not changed during thread's life-time.

See also
- ParentID Property 226
- Handle Property 222
- Name Property 225
- IsMainThread Property 223
- TEurekaThreadData Class 219

8.45.2.5 IsMainThread

Indicate whenever this thread is main thread.

Unit
EThreadsManager 136

Value
`True` for main thread, `False` for background threads.

Remarks
This property identifies main thread: that is a thread which TID (thread ID) equals to MainThreadID global variable.

There is only one thread in running application with this property set to `True`.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also
- Kind Property 224
- IsRTLThread Property 223
- IsWindowsThread Property 224
- TEurekaThreadData Class 219

8.45.2.6 IsRTLThread

Indicate whenever this thread is RTL thread.

Unit
EThreadsManager 136

Value
True for RTL threads, false for Windows threads.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

RTL thread is a thread created with BeginThread function or TThread class. Thread created with CreateThread function is considered to be non-RTL thread: it is a Windows thread.

This property is the opposite for IsWindowsThread 224 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

Note
Main thread is also considered to be a RTL thread.

See also

- Kind Property 224
- IsMainThread Property 223
- IsWindowsThread Property 224
- TEurekaThreadData Class 219

8.45.2.7 IsWindowsThread

Indicate whenever this thread is Windows thread.

Unit

EThreadsManager 136

Value

`True` for Windows threads, `False` for RTL threads.

Remarks

This property is the opposite for IsRTLThread 223 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- Kind Property 224
- IsRTLThread Property 223
- IsMainThread Property 223
- TEurekaThreadData Class 219

8.45.2.8 Kind

Thread kind (Windows/RTL thread).

Unit

EThreadsManager 136

Value

Indicates thread kind.

Remarks

Kind of the thread is detected on thread's creation. It is not changed during thread's life-time.

See also

- IsMainThread Property 223

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- IsRTLThread Property 223
- IsWindowsThread Property 224
- TEurekaThreadData Class 219

8.45.2.9 Name

Thread name (description).

Unit

EThreadsManager 136

Value

Optional thread name. Thread name (description) is an arbitrary string. Thead name does not have to be unique.

Remarks

This is an optional thread name, which describes the thread. Threads can be named to simplify debugging. Thread name is an arbitrary string which will be used in bug reports to describe thread. Does not have to be unique. Default value is an empty string.

The name of main thread is always 'Main thread'.

This is a dynamic property which can be changed while thread is running. A common (and recommended) practice is to set thread's name immediately after thread's start and do not change it later.

Use NameThread 496 or NameThreadForDebugging 496 functions to name a thread.

The name (description) of a thread can be set more than once; the most recently set value is used.

See also

- NameThread Function 496
- NameThreadForDebugging Function 496
- ID Property 223
- ParentID Property 226
- Handle Property 222
- IsMainThread Property 223
- TEurekaThreadData Class 219

8.45.2.10 Parameter

Usual thread function's argument.

Unit

EThreadsManager 136

Value

RAW pointer argument for StartingAddress 228 thread function.

Remarks

This property contains argument passed to thread function identified by StartingAddress 228 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- RealParameter Property 226
- StartingAddress Property 228
- StartingFunction Property 228

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- TEurekaThreadData Class 219

8.45.2.11 ParentID

Parent thread ID.

Unit

EThreadsManager 136

Value

ID of the thread which has created this thread. 0 for main thread.

Remarks

This is ID value for parent thread of this thread. A parent thread is a thread which invoked CreateThread/ BeginThread function for this thread.

Parent TID is detected on thread's creation. It is not changed during thread's life-time.

See also

- ID Property 223
- Handle Property 222
- Name Property 225
- IsMainThread Property 223
- TEurekaThreadData Class 219

8.45.2.12 RealParameter

Real thread function's argument.

Unit

EThreadsManager 136

Value

RAW pointer argument for RealStartingAddress 226 thread function.

Remarks

This property contains argument passed to thread function identified by RealStartingAddress 226 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- Parameter Property 225
- RealStartingAddress Property 226
- RealStartingFunction Property 227
- Thread Property 229
- ThreadClassName Property 229
- TEurekaThreadData Class 219

8.45.2.13 RealStartingAddress

Real thread function (pointer).

Unit

EThreadsManager 136

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
A pointer to thread function.

Remarks
This property points to real thread function. That is a thread function for CreateThread/BeginThread/ TThread.Execute.

A usual thread function is a function passed to `CreateThread` function. This value is stored in StartingAddress 228 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also
- StartingAddress Property 228
- CallerAddress Property 221
- RealStartingFunction Property 227
- RealParameter Property 226
- Thread Property 229
- ThreadClassName Property 229
- ThreadClassName Property 229
- TEurekaThreadData Class 219

8.45.2.14 RealStartingFunction

Real thread function (description).

Unit
EThreadsManager 136

Value
A human-readable description of RealStartingAddress 226 property.

Remarks
This value contains function name identified in RealStartingAddress 226 property. Function name is taken from debug information.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also
- StartingFunction Property 228
- CallerFunction Property 222
- RealStartingAddress Property 226
- RealParameter Property 226
- TEurekaThreadData Class 219

8.45.2.15 StackTop

Top of thread's start.

Unit
EThreadsManager 136

Value
A pointer to the beginning of the stack (top high address).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This value is used for call stack creation.

See also

- TEurekaThreadData Class 219

8.45.2.16 StartingAddress

Usual thread function (pointer).

Unit

EThreadsManager 136

Value

A pointer to usual thread function.

Remarks

This property points to usual thread function. That is a function passed directly to CreateThread - regardless of thread kind 224.

A real thread function is a function passed to `CreateThread/BeginThread/TThread.Execute` functions. This value is stored in RealStartingAddress 226 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

Note
This property equals to `RealStartingAddress` property for Windows threads and differs for RTL threads.

See also

- RealStartingAddress Property 226
- CallerAddress Property 221
- StartingFunction Property 228
- Parameter Property 225
- TEurekaThreadData Class 219

8.45.2.17 StartingFunction

Usual thread function (description).

Unit

EThreadsManager 136

Value

A human-readable description of StartingAddress 228 property.

Remarks

This value contains function name identified in StartingAddress 228 property. Function name is taken from debug information.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- RealStartingFunction Property 227

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- CallerFunction Property 222
- StartingAddress Property 228
- Parameter Property 225
- TEurekaThreadData Class 219

8.45.2.18 Thread

Thread object (class instance).

Unit
EThreadsManager 136

Value
A thread class instance for the thread. Typically this is a TThread instance. This value is `nil` for Windows 224 threads and for threads created with naked BeginThread function. This valus is not `nil` for `TThread` threads.

Remarks
This property stores actual thread object. Use typecasting to cast this object to appropriate thread class.

This value is detected on thread's creation. It is not changed during thread's life-time. However, properties of the thread object may change.

See also
- RealStartingAddress Property 226
- CallerAddress Property 221
- ThreadClassName Property 229
- TEurekaThreadData Class 219

8.45.2.19 ThreadClassName

Thread class name.

Unit
EThreadsManager 136

Value
Name of thread class. Usually this value is `TThread-descendant.ClassName`.

Remarks
This is class name of the class for the object in Thread 229 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also
- Thread Property 229
- TEurekaThreadData Class 219

8.46 TEurekaThreadDataList

List of threads.

Unit
EThreadsManager 136

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TEurekaThreadDataList = class(TCheckedInterfacedObject);
```

Remarks

Represents thread list returned by ThreadDataEnum 578 function.

Access to this list and to all items in the list does not require any locking/synchronization - because this list and all items are created upon `ThreadDataEnum` call.

This object is a "snapshot". It may not represent the actual state of threads in the application - because this state may be changed after taking the snapshot.

Properties 230	Properties
public	Public members

See also

- ThreadDataEnum Function 578
- TEurekaThreadData Class 219

8.46.1 Properties

Count 230	Count of threads in the list.
Items 231	Provides access to individual threads in the list.
ThreadData 231	Returns information about a particular thread.

See also

- TEurekaThreadDataList 229
- EThreadsManager 136

8.46.1.1 Count

Count of threads in the list.

Unit

EThreadsManager 136

Remarks

This property indicate count of threads in current list. Use this property to enumerate all threads via Items 231 property. If you're interested in a particular thread - use ThreadData 231 method.

See also

- Items property 231

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ThreadData Method 231
- ThreadDataEnum Function 578
- TEurekaThreadDataList Class 229

8.46.1.2 Items

Provides access to individual threads in the list.

Unit

EThreadsManager 136

Parameters

AIndex [in]

Index of a thread in the list. Index should be in 0.. Count 230-1 range.

Remarks

This is a default property for the `TEurekaThreadDataList` class.

This property can be used to enumerate all threads in the list. If you're interested in a particular thread - use ThreadData 231 method.

See also

- ThreadDataEnum Function 578
- ThreadData Method 231
- Count property 230
- TEurekaThreadDataList Class 229

8.46.1.3 ThreadData

Returns information about a particular thread.

Unit

EThreadsManager 136

Parameters

AThreadID [in]

TID (Thread ID) of a thread to get information.

Return value

Information about the `AThreadID` thread. Do not destroy the returned object. This value could be nil if `AThreadID` does not identify a valid running thread.

Remarks

Use this method to get information about a particular thread in the list.

There is no need to release the returned object.

There is no need to use locking - because the list and all items are created upon `ThreadDataEnum` call.

Any change for dynamic properties (such as Name, LastException, LastHTMLPage) will be ignored. Use ThreadData 577 function to modify thread's dynamic properties.

See also

- ThreadDataEnum Function 578
- ThreadData Function 577
- TEurekaThreadData Class 219

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- TEurekaThreadDataList Class 229

8.47 TEventLogDialog

Enter topic text here.

8.48 TExceptionManager

Exception manager class.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
TExceptionManager = class(TCheckedInterfacedObject, IInterface, IDLLExceptio
```

Remarks

This is class for exception manager object.

Exception manager is central control center for exceptions. Exception manager collects information about each exception, store information, invokes dialogs and sending, etc.

Caution!

Do not create instances of this class manually. Use ExceptionManager 396 or TExceptionManager.Instance 233 to access `TExceptionManager` class.

Methods 232	Methods
Properties 244	Properties
protected	Protected members
public	Public members

See also

- ExceptionManager function 396
- TExceptionManager.Instance class function 233

8.48.1 Methods

Instance 233	Returns singleton instance for `TExceptionManager` class.
ClearThreadExceptions 234	Clears last occurred exception in current thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Handle 234	Handles (processes) specified exception.
Info 237	Returns information about exception.
Notify 239	Notifies exception manager about exception.
ShowLastExceptionData 240	Handles last active exception (if any exists).
StandardEurekaError 241	Invokes EurekaLog without raising exception.
StandardEurekaNotify 242	Handle a specific exception.

See also

- TExceptionManager 232
- EExceptionManager 93

8.48.1.1 Instance

Returns singleton instance for `TExceptionManager` class.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
public
class function Instance: TExceptionManager 232;
```

Value

Instance of exception manager.

Remarks

This is global access method for exception manager. Use it to invoke routines of exception manager. Do not create instances of `TExceptionManager` class.

Exception manager instance is created and deleted automatically. You don't need to manually control life-time of this object.

See TExceptionManager 232 for more information.

Note

This method is the same as ExceptionManager 396 function.

See also

- ExceptionManager function 396
- TExceptionManager Class 232

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.48.1.2 ClearThreadExceptions

Clears last occurred exception in current thread.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
public
procedure ClearThreadExceptions;
   overload;
```

Remarks

This function unregisters last occured exception in current thread. This function deletes exception information about last occured exception in current thread.

After calling this function:

- LastThreadException 244 will return `nil`.
- ShowLastExceptionData 240 will do nothing (return srNoExceptionInfo 726 code).
- Corresponding TEurekaExceptionInfo 180 object will be destroyed.

Note

This method affects current thread only. Other threads are unaffected. Calling this function is thread-safe.

See also

- ShowLastExceptionData method 240
- LastThreadException property 244
- TEurekaExceptionInfo class 180
- TExceptionManager Class 232

8.48.1.3 Handle

Handles (processes) specified exception.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
public
function Handle(
   const AInfo: TEurekaExceptionInfo 180;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse 713; overload;

function Handle(
   const AInfo: TExceptionData;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): TResponse[713]; overload;

function Handle(
   const AInfo: Pointer;
   const AAddr: Pointer = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[713]; overload;

function Handle(
   const AInfo: TObject;
   const AAddr: Pointer = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[713]; overload;

function Handle(
   const AInfo: Windows.TExceptionRecord;
   const AContext: PContext = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[713]; overload;

function Handle(
   const AInfo: TExceptionPointers;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[713]; overload;
```

Parameters

AInfo [in]

Exception object, exception information, exception data, or any other information which identifies exception (there are multiple overloads variants of the same `Info` method). If this value is not specified (`nil`) - then method will return srNoExceptionInfo[726].

AAddr [in, optional]

Exception address. Typically this is ExceptAddr value. This argument is present not in all overloaded methods.

AContext [in, optional]

Pointer to low-level CPU information. This information will be used for creation of new information. It's not used for identification purposes.

AHandled [in, optional]

`True` - this exception is handled. `False` - this exception is unhandled.

Exception information will be modified with this value. Handled/Unhandled sign is used for two different kinds of exception processing. "Handled" exceptions are not processed by default.

ARespectActivation [in, optional]

`True` (default) - check the current status of EurekaLog. If no EurekaLog is enabled - return `srRestart`, otherwise (EurekaLog is enabled) - do the usual work.

`False` - always process the exceptions even if EurekaLog is not active.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Result of exception processing. This record hold multiple fields with information - code, messages, etc. See TResponse 713 for more information.

Remarks

This method performs processing of exception - that is: dialog, bug report saving, sending.

This function is a wrapper for ProcessException function, except it also updates exception manager state:

1. This method invokes OnExceptionNotify 14 event.
2. Exception filters are launched (if present).
3. Statistics for auto-restart feature is calculated.
4. ProcessException function is called to perform actual processing.
5. This method is also handles all exceptions during processing, converting them to `TResponse` value.

Important!

It's recommended to use `Handle` method of exception manager instead of calling `ProcessException` directly.

Important!

`Handle` method will process only already raised exceptions (i.e. with actual call stack). `Handle` method will return srNoExceptionInfo 726 for non-raised exceptions. For example:

Code (Delphi)

```
E := Exception.Create('My custom exception');
         ExceptionManager.Handle(E); // - will do nothing, return srNo
```

Code (Delphi)

```
CallStack := EurekaCallStackClass.Create(EurekaCallStackClass.Caller);
         try
          EI.CallStack := CallStack;
         finally
          FreeAndNil(CallStack);
         end;
         ExceptionManager.Handle(E); // - will do the usual processing
```

Examples

Code (Delphi)

```
try
// ... your code ...
except
   ExceptionManager.Handle(ExceptObject, ExceptAddr);
end;
```

Code (Delphi)

```
Thread := TMyThread.Create(False);
try
   Thread.WaitFor;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   ExceptionManager.Handle(Thread.FatalException);
finally
   FreeAndNil(Thread);
end;
```

See also

- StandardEurekaNotify Method 242
- TResponse Record 713
- Info Method 237
- Notify Method 239
- LastThreadException Property 244
- ProcessException Function
- OnExceptionNotify Event 14
- TExceptionManager Class 232

8.48.1.4 Info

Returns information about exception.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
public
function Info(
   const AInfo: Pointer;
   const AAddr: Pointer = nil;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 180; overload;

function Info(
   const AInfo: TObject;
   const AAddr: Pointer = nil;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 180; overload;

function Info(
   const AInfo: TObject;
   const AExceptionRecord: Windows.TExceptionRecord;
   const AContext: PContext = nil;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 180; overload;

function Info(
   const AInfo: TObject;
   const AEP: TExceptionPointers;
   const AForcedNew: Boolean = False;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    const ARespectActivation: Boolean = True;
    const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo; overload;

function Info(
    var AInfo: TExceptionData;
    const AForcedNew: Boolean = False;
    const ARespectActivation: Boolean = True;
    const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo; overload;

function Info(
    const AInfo: Windows.TExceptionRecord;
    const AContext: PContext = nil;
    const AForcedNew: Boolean = False;
    const ARespectActivation: Boolean = True;
    const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo; overload;

function Info(
    const AInfo: TExceptionPointers;
    const AForcedNew: Boolean = False;
    const ARespectActivation: Boolean = True;
    const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo; overload;
```

Parameters

AInfo [in]

Exception object, exception information, exception data, or any other information which identifies exception (there are multiple overloads variants of the same `Info` method). If this value is not specified (`nil`) - then `nil` will be returned from the method.

AAddr [in, optional]

Exception address. Typically this is ExceptAddr value. This argument is present not in all overloaded methods.

AContext [in, optional]

Pointer to low-level CPU information. This information will be used for creation of new information. It's not used for identification purposes.

AForcedNew [in, optional]

`False` (default) - standard behaviour: try to locate already created info, reate new information only when no existing info is found.

`True` - always create new information instance. It's recommended don't use this value unless you're 100% sure that your exception is new (just created). Value of `True` is used by exception hooks.

ARespectActivation [in, optional]

`True` (default) - check the current status of EurekaLog. If no EurekaLog is enabled - return `nil`, otherwise (EurekaLog is enabled) - do the usual work.

`False` - always retrieve information even if EurekaLog is not active.

Return value

Information about specified exception or `nil` if information doesn't exist.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Notify method 239
- Handle method 234
- LastThreadException property 244
- TExceptionManager Class 232

8.48.1.5 Notify

Notifies exception manager about exception.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
protected
procedure Notify(
   const AInfo: TEurekaExceptionInfo 180;
   const ARespectActivation: Boolean = True;
   const AExternalThread: Boolean = False
); overload;

procedure Notify(
   const AInfo: TExceptionData;
   const ARespectActivation: Boolean = True
); overload;

procedure Notify(
   const AInfo: Pointer;
   const AAddr: Pointer = nil;
   const ARespectActivation: Boolean = True
); overload;

procedure Notify(
   const AInfo: TObject;
   const AAddr: Pointer = nil;
   const ARespectActivation: Boolean = True
); overload;

procedure Notify(
   const AInfo: Windows.TExceptionRecord;
   const AContext: PContext = nil;
   const ARespectActivation: Boolean = True
); overload;

procedure Notify(
   const AInfo: TExceptionPointers;
   const ARespectActivation: Boolean = True
); overload;
```

Parameters

AInfo [in]

Exception object, exception information, exception data, or any other information which identifies exception (there are multiple overloads variants of the same `Info` method). If this value is not specified

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

(`nil`) - then `nil` will be returned from the method.

AAddr [in, optional]
Exception address. Typically this is ExceptAddr value. This argument is present not in all overloaded methods.

AContext [in, optional]
Pointer to low-level CPU information. This information will be used for creation of new information. It's not used for identification purposes.

ARespectActivation [in, optional]
`True` (default) - check the current status of EurekaLog. If no EurekaLog is enabled - return `nil`, otherwise (EurekaLog is enabled) - do the usual work.

`False` - always retrieve information even if EurekaLog is not active.

Return value
Information about specified exception or `nil` if information can not be created.

Remarks

Warning
Do not call this method directly! It's called only from hooks to notify exception manager about exceptions.

1. First, this method retrieves actual exception information via `Info` method. If no information exists - it will be created.
2. The method updates counters (exception count 244)
3. Updates last exception 244.
4. Build chained exceptions chain.
5. Removes obsolete information blocks (deleted/not active exceptions).
6. Invokes OnRaise 12 event.

See also
- Info method 237
- Handle method 234
- LastThreadException property 244
- OnRaise event 12
- TExceptionManager Class 232

8.48.1.6 ShowLastExceptionData

Handles last active exception (if any exists).

Unit
EExceptionManager 93

Syntax

Code (Delphi)

```
public
function ShowLastExceptionData: TResponse 713;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value
Result of exception processing.

Remarks
This is the same as:

Handle 234(LastThreadException 244);

If there is no active exception - srNoExceptionInfo 726 code will be returned.

This action usually means displaying error dialog, generating/saving bug report, sending report, etc.

This method is called from installed hooks. Hooks for such methods as TApplication.ShowException.

Tip
This method is EurekaLog-equivalent of TApplication.HandleException method.

See also
- Handle method 234
- LastThreadException property 244
- ClearThreadExceptions method 234
- StandardEurekaNotify method 242
- StandardEurekaError method 241
- TExceptionManager Class 232

8.48.1.7 StandardEurekaError

Invokes EurekaLog without raising exception.

Unit
EExceptionManager 93

Syntax

Code (Delphi)

```
public
function StandardEurekaError(
   const Error: String;
   const AOptions: TEurekaModuleOptions 216 = nil
): TResponse 713;
```

Parameters
Error [in]
Error message to display in dialogs and report.

AOptions [in, optional]
EurekaLog options for this operation. Most important options are: csoShowRTLThreads, csoShowWindowsThreads and csoShowELThreads properties.

Return value
Result of processing. See Handle 234 and TResponse 713 for more information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function creates exception of EEurekaLogGeneralError class and passes it to Handle 234 method.

Use this method to invoke EurekaLog (i.e. show dialog, save report, send report) without actually raising exception. You can also setup exception filters or event handlers to alter processing behaviour of this "exception". Use `EEurekaLogGeneralError` class from `EBase` unit to differ this exception from typical exceptions.

The advantage of this function over usual raising method is that no exception will be raised at all (even though exception object will be created). Thus, you will save some performance and you also will not be disturbed by exception notification dialog from debugger.

Examples

Code (Delphi)

```
ExceptionManager.StandardEurekaError('File does not exist, unable to cor
```

See also

- StandardEurekaNotify method 242
- ShowLastExceptionData method 240
- Info method 237
- Handle method 234
- TExceptionManager Class 232

8.48.1.8 StandardEurekaNotify

Handle a specific exception.

Unit

EExceptionManager 93

Syntax

Code (Delphi)

```
public
function StandardEurekaNotify(
   const Obj: TObject;
   const Addr: Pointer = nil;
   const AOptions: TEurekaModuleOptions 216 = nil
): TResponse 713;
```

Parameters

Obj [in]
Exception object. Can be value from ExceptObject function.

Addr [in, optional]
Exception address. Can be value from ExceptAddr function.

AOptions [in, optional]
EurekaLog options for this operation. Most important options are: csoShowRTLThreads, csoShowWindowsThreads and csoShowELThreads properties.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Result of exception's processing. See Handle 234 and TResponse 713 for more information.

Remarks

This is a simple combination of Info 237 and Handle 234 methods. This method is a tool for adding EurekaLog to unsupported applications types. You can call this method from your custom exception handler to invoke EurekaLog's processing.

This method is similar to ShowLastExceptionData 240, except it allows to specify exception object.

Tip

The diffrence between this method and the Handle 234 method is this: `Handle` method will do nothing for non-raised exceptions. `StandardEurekaNotify` will create call stack for non-raised exceptions.

Examples

Code (Delphi)

```
try
   // ... your code ...
except
   ExceptionManager.StandardEurekaNotify(ExceptObject, ExceptAddr);
end;
```

Code (Delphi)

```
Thread := TMyThread.Create(False);
try
   Thread.WaitFor;
   ExceptionManager.StandardEurekaNotify(Thread.FatalException);
finally
   FreeAndNil(Thread);
end;
```

Code (Delphi)

```
E := EMyException.Create('My Error Message');
try
   ExceptionManager.StandardEurekaNotify(E);
finally
   FreeAndNil(E);
end;
```

See also

- StandardEurekaError method 241
- ShowLastExceptionData method 240
- Info method 237
- Handle method 234
- LastThreadException property 244
- ClearThreadExceptions method 234
- TExceptionManager Class 232

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.48.2 Properties

ExceptionCount 244	Count of all exceptions.
LastThreadException 244	Returns last occurred exception in current thread.
Mode 245	Indicate work mode of exception manager.
UnhandledExceptionCount 245	Count of all processed exceptions.

See also

- TExceptionManager 232
- EExceptionManager 93

8.48.2.1 ExceptionCount

Count of all exceptions.

Unit

EExceptionManager 93

Value

Count of all catched exceptions since application start. This value is >= 0 and always increasing as time progresses. This value never decreases.

Remarks

Every time when exception manager founds new exception - this value is increased by 1. Actual counter is also saved to exception's information (see TEurekaExceptionInfo 180.ExceptionNumber 193).

This value holds count of discovered exceptions regardless of their kind (handled or not). If you want to get number of actually processed exceptions - use UnhandledExceptionCount 245 property.

Tip
This value does NOT represent count of active exceptions.

See also

- UnhandledExceptionCount property 245
- TEurekaExceptionInfo.ExceptionNumber property 193
- TEurekaExceptionInfo class 180
- TExceptionManager Class 232

8.48.2.2 LastThreadException

Returns last occurred exception in current thread.

Unit

EExceptionManager 93

Value

Information about most recent raised exception for active thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Returns information about last raised exception in current thread. You can use this property to obtain information about last exception.

If you want to get information about arbitrary exception object (it's not needed to be the last raised, it even may be already handled and processed) - then call Info 237 method.

Tip
This method is EurekaLog-equivalent of a ExceptObject function.

Note
This method returns information about current thread only. It won't return information about exceptions in other threads. Calling this method is thread-safe.

See also

- ShowLastExceptionData method 240
- ClearThreadExceptions method 234
- TEurekaExceptionInfo class 180
- Handle method 234
- TExceptionManager Class 232

8.48.2.3 Mode

Indicate work mode of exception manager.

Unit

EExceptionManager 93

Value

State of additional hooks.

Remarks

Work mode indicate what kind of additional hooks were installed by exception manager. Additional hooks are required for some special features to work. This includes "Catch handled exceptions" and "Handle safecall exceptions" options. This mode doesn't indicate state of primary hooks.

See also

- TExceptionNotifyMode type 684
- TExceptionManager Class 232

8.48.2.4 UnhandledExceptionCount

Count of all processed exceptions.

Unit

EExceptionManager 93

Value

Count of all exceptions that were processed since application start. This value is >= 0 and always increasing as time progresses. This value never decreases.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Every time when exception manager handles exception (i.e. shows dialog, saves bug report, etc.) - this value is increased by 1. Actual counter is also saved to exception's information (see TEurekaExceptionInfo 180 .UnhandledExceptionNumber 208).

This value holds count of processed exceptions, it's always **less or equal** to ExceptionCount 244 property.

Tip
If you set your application to terminate or restart after the very first unhandled exception - then this value will be 0 all the time. And it will be 1 during exception processing, but you will not see 2 here.

Tip
This value does NOT represent count of active exceptions.

See also

- ExceptionCount property 244
- Handle method 234
- TEurekaExceptionInfo.UnhandledExceptionNumber property 208
- TEurekaExceptionInfo class 180
- TExceptionManager Class 232

8.49 TFreezeThread

Base class for freeze detection threads.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
TFreezeThread = class(TThreadEx 255);
```

Remarks

This is an abstract class to be used as base class when implementing background freeze detection thread. Override the Execute 248 method and implement your own freeze detection logic. Call RaiseFreezeException 499 once you detect freeze/hang/deadlock.

Note
FreezeThreadClass 756 variable determinates which class EurekaLog should use. Use InitCheckFreeze 436 to re-initialize freeze detection if it is already running.

Methods 247	Methods
Properties 248	Properties
protected	Protected members

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

public	Public members

See also

- TMainThreadFreezeDetectionThread Class 254
- TWCTFreezeDetectionThread Function 265
- TFreezeThreadClass Type 686
- FreezeThreadClass Variable 756
- InitCheckFreeze Function 436

8.49.1 Methods

ChecksPaused 247	Indicates whether freeze checks are disabled.
Execute 248	Implements actual freeze detection code.

See also

- TFreezeThread 246
- EFreeze 96

8.49.1.1 ChecksPaused

Indicates whether freeze checks are disabled.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
protected
function ChecksPaused: Boolean;
```

Return value

`True` - freeze checks are disabled; `False` - freeze checks are enabled.

Remarks

The thread's Execute 248 method and any methods that `Execute` calls should check `ChecksPaused` periodically and do nothing when it's `True`. The PauseFreezeCheck 497 procedure sets the `ChecksPaused` property to `True`.

See also

- PauseFreezeCheck Function 497
- Terminated Property
- TFreezeThread Class 246

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.49.1.2 Execute

Implements actual freeze detection code.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
protected
procedure Execute;
   override;
```

Remarks

Override `Execute` and insert the code that should be executed when the thread runs. `Execute` is responsible for checking the value of the Terminated property to determine if the thread needs to exit, and for checking the value of ChecksPaused 247 property to determinate if freeze detection checks are disabled.

Important!

Do not forget to call inherited `Execute` implementation in your own overridden implementations of `Execute`.

See also

- TFreezeThread Class 246

8.49.2 Properties

FreezeTimeout 248	Freeze timeout from InitCheckFreeze 436 .
TargetThreadHandle 249	Thread handle from InitCheckFreeze 436 .
TargetThreadID 249	Thread ID from InitCheckFreeze 436 .
TargetWindow 249	Window handle from InitCheckFreeze 436 .

See also

- TFreezeThread 246
- EFreeze 96

8.49.2.1 FreezeTimeout

Freeze timeout from InitCheckFreeze 436 .

Unit

EFreeze 96

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- InitCheckFreeze Function 436
- TFreezeThread Class 246

8.49.2.2 TargetThreadHandle

Thread handle from InitCheckFreeze 436.

Unit

EFreeze 96

See also

- InitCheckFreeze Function 436
- TFreezeThread Class 246

8.49.2.3 TargetThreadID

Thread ID from InitCheckFreeze 436.

Unit

EFreeze 96

See also

- InitCheckFreeze Function 436
- TFreezeThread Class 246

8.49.2.4 TargetWindow

Window handle from InitCheckFreeze 436.

Unit

EFreeze 96

See also

- InitCheckFreeze Function 436
- TFreezeThread Class 246

8.50 TGenericStackList

Common stack tracing class.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
TGenericStackList = class(TEurekaBaseStackList 172);
```

Remarks

Use this class for building call stack by any method, except EurekaLog.

Warning

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Do not use this class for EurekaLog stack tracing.

Tip

Avoid using classes directly. Use EurekaCallStackClass 755/ GetTracer 430/GetTracerClass 431.

Methods 250	Methods
Properties 251	Properties
public	Public members

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerWindows);
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TEurekaStackListV7 class 218
- TEurekaStackListV6 class 217

8.50.1 Methods

Create 250	Creates call stack class with specified tracer.

See also

- TGenericStackList 249
- ECallStack 59

8.50.1.1 Create

Creates call stack class with specified tracer.

Unit

ECallStack 59

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
public
constructor Create(
   const ATracerNo: Integer = TracerUndefined
); reintroduce;
```

Parameters

ATracerNo [in]

Number of stack tracer. Numbers are listed as Tracer*XYZ* constants in EStackTracing 133 unit. Some examples are: TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesJCL TracerLeaksFrame TracerWindows Any other Tracer*XYZ* value...

Remarks

This class can be used to build call stack with arbitrary stack tracing method (except EurekaLog tracing methods).

Warning

Do not use this class for EurekaLog stack tracing.

Use Build 178 method to trace stack after creation.

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerWindows);
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TGenericStackList Class 249

8.50.2 Properties

Tracer 252	Indicates stack tracer used to build call stack.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TGenericStackList 249
- ECallStack 59

8.50.2.1 Tracer

Indicates stack tracer used to build call stack.

Unit

ECallStack 59

Value

One of Tracer*XYZ* values from EStackTracing unit: TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesJCL TracerLeaksFrame TracerWindows Any other Tracer*XYZ* value...

Remarks

This property holds `ATracerNo` argument, which was used in `Create` constructor.

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerWindows);
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);


      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TGenericStackList Class 249

8.51 TJSONLogBuilder

JSON EurekaLog 7+ log builder.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TJSONLogBuilder = class(TBaseLogBuilder 152);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This is log builder class for EurekaLog 7 and above. It is used to save JSON bug reports.

8.52 TLogBuilder

Default EurekaLog 7 log builder.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TLogBuilder = class(TBaseLogBuilder 152);
```

Remarks

This is default log builder class for EurekaLog 7 and above. It's used as default value for LogBuilderClass 757 global variable.

There are 3 more build-in log builder classes: TXMLLogBuilder 265, TJSONLogBuilder 252, and TDummyLogBuilder 159.

This class creates such report as:

Code

```
EurekaLog 7.0.5.41 RC 13

Exception:
------------------------------
   2.2 Address: 005CBFB4
   2.5 Type : Exception
   2.6 Message: Error Message.
   2.7 ID : 8F800000
   2.11 Sent : 0

User:
----------------------------------
   3.2 Name : randomclear@gmail.com
   3.3 Email:

Steps to reproduce:
------------
   8.1 Text:


Call Stack Information:
-------------------------------------------------------------------------------
|Methods |Details|Stack |Address |Module |Offset |Unit |Class |Procedure
-------------------------------------------------------------------------------
|*Exception Thread: ID=9984; Parent=0; Priority=0 |
|Class=; Name=MAIN |
|DeadLock=0; Wait Chain= |
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
|Comment= |
|-------------------------------------------------------------------------
|7FFFFFFE|04 |00000000|005CBFB4|Project4.exe|001CBFB4|Unit4 |TForm4 |But
|00000020|03 |0018FEF0|74658A61|user32.dll |00008A61|user32 | |DispatchM
|00000020|03 |0018FF00|005C16FF|Project4.exe|001C16FF|Forms |TApplicatio
|00000020|03 |0018FF1C|005C1742|Project4.exe|001C1742|Forms |TApplicatio
...
etc.
```

8.53 TMainThreadFreezeDetectionThread

Thread class to detects hangs in UI thread.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
TMainThreadFreezeDetectionThread = class(TFreezeThread 246);
```

Remarks

This thread uses SendMessageTimeout function to detect if main thread is responding on messages.

Note

This freeze detection thread class can be used on any system.

EurekaLog uses this class by default on pre-Vista systems (Windows XP and earlier).

8.54 TMessageBoxDetailedDialog

Enter topic text here.

8.55 TMessageBoxDialog

Enter topic text here.

8.56 TMSClassicDialog

Enter topic text here.

8.57 TNullDialog

Enter topic text here.

8.58 TRTLHandlerDialog

Enter topic text here.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.59 TThreadEx

Abstract class to create separate threads of execution.

Unit

EBase 57

Syntax

Code (Delphi)

```
TThreadEx = class(TThreadHook);
```

Remarks

`TThreadEx` is EurekaLog's replacement for TThread class. `TThreadEx` offers two additional arguments in constructor, which allows you to control thread's name and EurekaLog's state in this thread.

Create a descendant of `TThreadEx` to represent an execution thread in a multithreaded application. Each new instance of a `TThreadEx` descendant is a new thread of execution. Multiple instances of a `TThreadEx` derived class make an application multithreaded.

When an application is run, it is loaded into memory ready for execution. At this point it becomes a process containing one or more threads that contain the data, code and other system resources for the program. A thread executes one part of an application and is allocated CPU time by the operating system. All threads of a process share the same address space and can access the process's global variables.

Define the thread object's Execute 261 method by inserting the code that should execute when the thread is executed.

Important!

You **must** call the inherited implementation in your overridden Execute 261 method.

Methods 258	Methods
Properties 263	Properties
protected	Protected members
public	Public members

Examples

Code (Delphi)

```
type
   TMyThread = class(TThreadEx)
   protected
      procedure Execute; override;
   end;

procedure TMyThread.Execute;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
begin
   inherited; // - this is required

   // ... your code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TMyThread;
   E: TObject;
begin
   // Create thread
   Thread := TMyThread.Create('Thread Name');
   try

      // Wait for thread's completion.
      // This wait can be implemented in any other way.
      // E.g. you can assign OnTerminate handler;
      // or you can PostMessage from thread to main thread.
      Thread.WaitFor;

      // Analyze thread completion.
      // Re-raise any thread error in current thread.
      // You should do this only after the thread has finished.
      E := Thread.FatalException;
      if Assigned(E) then
      begin
         // clear FatalException property
         PPointer(@Thread.FatalException)^ := nil;
         raise E;
      end;

   finally
      FreeAndNil(Thread);
   end;
end;
```

Threads launched with `TThreadEx` class will be EurekaLog-enabled by default. You can supply optional `Boolean` argument for `TThreadEx`'s constructor to disable EurekaLog in thread, for example:

Code (Delphi)

```
Thread := TMyThread.Create('Thread Name', False { disable EurekaLog in t
```

Of course, you can use the same approach while supplying `Suspended` argument, for example:

Code (Delphi)

```
Thread := TMyThread.Create(
   False { create thread running },
   'Thread Name',
   False { disable EurekaLog in thread } );

Thread := TMyThread.Create(
   True { create thread suspended},
   'Thread Name',
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   True { enable EurekaLog in thread } );
```

TThreadEx class also supports anonymous threads:

Code (Delphi)

```
type
   TForm1 = class(TForm)
      ...
   private
      procedure HandleThreadException(Sender: TObject);
   end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TThreadEx;
begin
   Thread := TThreadEx.CreateAnonymousThread(
      procedure
      begin
         raise Exception.Create('Test');
      end,
      'My thread');
   Thread.OnTerminate := Self.HandleThreadException;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
   // All anonymous threads are marked with FreeOnTerminate by default
end;

procedure TForm1.HandleThreadException(Sender: TObject);
var
   Thread: TThread;
begin
   Thread := (Sender as TThread);
   if Assigned(Thread.FatalException) then
      HandleException(Thread.FatalException, False, atThread);
end;
```

Since analyzing Thread.FatalException property or creating OnTerminate handler just for handling exceptions in thread is a lot of work - TThreadEx class presents a new property to handle exceptions automatically:

Code (Delphi)

```
type
   TMyThread = class(TThreadEx)
   protected
      procedure Execute; override;
   end;

procedure TMyThread.Execute;
begin
   inherited; // - this is required

   // ... your code ...
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TMyThread;
begin
   Thread := TMyThread.Create(True, 'My thread');
   Thread.AutoHandleException := True;
   Thread.FreeOnTerminate := True;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
end;
```

See also

- BeginThreadEx Function 325
- SetEurekaLogStateInThread Function
- NameThread Function 496
- HandleException Function 433

8.59.1 Methods

Create 258	Creates an instance of a thread object.
Destroy 260	No description available
Execute 261	No description available

See also

- TThreadEx 255
- EBase 57

8.59.1.1 Create

Creates an instance of a thread object.

Unit

EBase 57

Syntax

Code (Delphi)

```
public
constructor Create;
   overload;

constructor Create(
   CreateSuspended: Boolean
); overload;

constructor Create(
   const AName: String;
   const AELEnabled: Boolean = Boolean(2
)); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
constructor Create(
   CreateSuspended: Boolean;
   const AName: String;
   const AELEnabled: Boolean = Boolean(2
)); overload;
```

Parameters

CreateSuspended [in, opt]

If CreateSuspended is False, Execute 261 is called immediately after the constructor. If CreateSuspended is True, Execute 261 is not called until after the Start method is called. Default is False.

ReservedStackSize [in, opt]

The initial size of the stack, in bytes. The system rounds this value to the nearest page. If this parameter is zero, the new thread uses the default size for the executable. Default is 0.

AName [in, opt]

Defines thread's name. Thread name is arbitrary text string which can contain any combination of characters. Thread name will be visible in IDE's debugger. It also will be used by EurekaLog in bug reports. Default is empty name (no name; TID will be used to identify this thread).

AELEnabled [in, opt]

Defines EurekaLog status in created thread. This argument is ignored if you compile your application without EurekaLog. Default is the same state as current thread.

Remarks

Call Create to create a thread in an application.

Examples

Code (Delphi)

```
// Execute thread immediately:
Thread := TMyThread.Create;
```

Code (Delphi)

```
// Create thread suspended:
Thread := TMyThread.Create(True);

// ...

// Start thread's execution
Thread.Start;
```

Code (Delphi)

```
// Name and execute thread immediately:
Thread := TMyThread.Create('My background thread');
```

Code (Delphi)

```
// Create named thread suspended:
Thread := TMyThread.Create(True, 'My background thread');
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
// ...

// Start thread's execution
Thread.Start;
```

Threads launched with `TThreadEx` class will be EurekaLog-enabled by default. You can supply optional `Boolean` argument for `TThreadEx`'s constructor to disable EurekaLog in thread, for example:

Code (Delphi)

```
Thread := TMyThread.Create('Thread Name', False { disable EurekaLog in t
```

Of course, you can use the same approach while supplying `Suspended` argument, for example:

Code (Delphi)

```
Thread := TMyThread.Create(
   False { create thread running },
   'Thread Name',
   False { disable EurekaLog in thread } );

Thread := TMyThread.Create(
   True { create thread suspended},
   'Thread Name',
   True { enable EurekaLog in thread } );
```

See also

- SetEurekaLogStateInThread Function
- NameThread Function 496
- HandleException Function 433
- TThreadEx Class 255

8.59.1.2 Destroy

Unit

EBase 57

Syntax

Code (Delphi)

```
public
destructor Destroy;
   override;
```

Remarks

See also

- TThreadEx Class 255

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.59.1.3 Execute

Defines thread's execution.

Unit

EBase 57

Syntax

Code (Delphi)

```
protected
procedure Execute;
   reintroduce;
   virtual;
```

Remarks

Override `Execute` and insert the code that should be executed when the thread runs. `Execute` is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Important!

You **must** call the inherited implementation in your overridden `Execute` method.

A thread executes when Create 258 is called if `CreateSuspended` set to `False`, or when Start is first called after the thread is created if `CreateSuspended` set to `True`.

Examples

Code (Delphi)

```
type
   TMyThread = class(TThreadEx)
   protected
      procedure Execute; override;
   end;

procedure TMyThread.Execute;
begin
   inherited; // - this is required

   // ... your code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TMyThread;
   E: TObject;
begin
   // Create thread
   Thread := TMyThread.Create('Thread Name');
   try

      // Wait for thread's completion.
      // This wait can be implemented in any other way.
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
      // E.g. you can assign OnTerminate handler;
      // or you can PostMessage from thread to main thread.
      Thread.WaitFor;

      // Analyze thread completion.
      // Re-raise any thread error in current thread.
      // You should do this only after the thread has finished.
      E := Thread.FatalException;
      if Assigned(E) then
      begin
         // clear FatalException property
         PPointer(@Thread.FatalException)^ := nil;
         raise E;
      end;

   finally
      FreeAndNil(Thread);
   end;
end;
```

Since analyzing `FatalException` property or creating `OnTerminate` handler just for handling exceptions in thread is a lot of work - `TThreadEx` class presents a new property to handle exceptions automatically:

Code (Delphi)

```
type
   TMyThread = class(TThreadEx)
   protected
      procedure Execute; override;
   end;

procedure TMyThread.Execute;
begin
   inherited; // - this is required

   // ... your code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TMyThread;
begin
   Thread := TMyThread.Create(True, 'My thread');
   Thread.AutoHandleException := True;
   Thread.FreeOnTerminate := True;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
end;
```

See also

- SetEurekaLogStateInThread Function
- NameThread Function 496
- HandleException Function 433
- TThreadEx Class 255

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.59.2 Properties

AutoHandleException 263	No description available

See also

- TThreadEx 255
- EBase 57

8.59.2.1 AutoHandleException

Allows you to enable automatic thread's exception handling.

Unit

EBase 57

Value

`True` - automatically invoke EurekaLog for exceptions in thread. `False` (default) - do not alter standard behaviour; store thread exception in FatalException property; you **must** analyze this exception manually.

Remarks

By default this property is set to `False`. This means that `TThreadEx` class will behave in the same way as TThread class - it will store thread's fatal exception into FatalException property, which **must** be analyzed by caller thread. For example:

Code (Delphi)

```
// Create thread
Thread := TMyThread.Create('Thread Name');
try

   // Wait for thread's completion.
   // This wait can be implemented in any other way.
   // E.g. you can assign OnTerminate handler;
   // or you can PostMessage from thread to main thread.
   Thread.WaitFor;

   // Analyze thread completion.
   // Re-raise any thread error in current thread.
   // You should do this only after the thread has finished.
   E := Thread.FatalException;
   if Assigned(E) then
   begin
      // clear FatalException property
      PPointer(@Thread.FatalException)^ := nil;
      raise E;
   end;

finally
   FreeAndNil(Thread);
end;
```

or:

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
type
   TForm1 = class(TForm)
      ...
   private
      procedure HandleThreadException(Sender: TObject);
   end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TThreadEx;
begin
   Thread := TThreadEx.Create(True);
   Thread.OnTerminate := Self.HandleThreadException;
   Thread.FreeOnTerminate := True;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
end;

procedure TForm1.HandleThreadException(Sender: TObject);
var
   Thread: TThread;
begin
   Thread := (Sender as TThread);
   if Assigned(Thread.FatalException) then
      HandleException(Thread.FatalException, False, atThread);
end;
```

When `AutoHandleException` property is set to `True` - EurekaLog will call HandleException 433 function automatically for thread's fatal exceptions.

Code (Delphi)

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TThreadEx;
begin
   Thread := TThreadEx.Create(True);
   Thread.AutoHandleException := True;
   Thread.FreeOnTerminate := True;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
end;
```

Please note that even when `AutoHandleException` is `True` - FatalException property will still be filled with exception.

See also

- SetEurekaLogStateInThread Function
- NameThread Function 496
- HandleException Function 433
- TThreadEx Class 255

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.60 TWCTFreezeDetectionThread

Thread class to detects deadlocks between threads.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
TWCTFreezeDetectionThread = class(TFreezeThread 246);
```

Remarks

This thread uses Wait Chain Traversal function to detect deadlocks between two or more EurekaLog-enabled threads.

Note

This freeze detection thread class can be used on Vista+ systems only.

EurekaLog uses this class by default on Vista+ systems (Windows Vista and later).

8.61 TWebDialog

Enter topic text here.

8.62 TWebDialog

Enter topic text here.

8.63 TWebDialog

Enter topic text here.

8.64 TWERDialog

Enter topic text here.

8.65 TXMLLogBuilder

XML EurekaLog 7+ log builder.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TXMLLogBuilder = class(TBaseLogBuilder 152);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This is log builder class for EurekaLog 7 and above. It is used to save XML bug reports.

8.66 TZipFile

Class for creating and reading .ZIP files.

Unit

EZip 144

Syntax

Code (Delphi)

```
TZipFile = class(TObject);
```

Remarks

Methods 266	Methods
Properties 274	Properties
private	Private members
public	Public members

8.66.1 Methods

ExtractZipFile 267	Extract a ZipFile
IsValid 267	Verifies .ZIP file
ZipDirectoryContents 268	Zip the contents of a directory
Create 268	Create a TZipFile
Destroy 269	No description available
Add 269	Add a file to the ZIP file
Close 271	No description available
Extract 271	Extract a single file
ExtractAll 272	Extract All files
IndexOf 272	Translate from FileName to index in ZIP

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	Central Header
Open 273	Opens a ZIP file for reading or writing.
Read 273	Read a file from arcive to an array of Bytes

See also

- TZipFile 266
- EZip 144

8.66.1.1 ExtractZipFile

Extract a ZipFile

Unit

EZip 144

Syntax

Code (Delphi)

```
public
class function ExtractZipFile(
   const ZipFileName, Path: String;
   const APassword: String = ''
): String; static;
```

Parameters

ZipFileName
File name of the ZIP file

Path
Path to extract to disk

See also

- TZipFile Class 266

8.66.1.2 IsValid

Verifies .ZIP file

Unit

EZip 144

Syntax

Code (Delphi)

```
public
class function IsValid(
   const ZipFileName: String
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): Boolean; static;
```

Parameters

ZipFileName
Path to Zip File

Return value

Is the .ZIP file valid

See also

- TZipFile Class 266

8.66.1.3 ZipDirectoryContents

Zip the contents of a directory

Unit

EZip 144

Syntax

Code (Delphi)

```
public
class procedure ZipDirectoryContents(
   const ZipFileName, Path: String;
   Compression: TZipCompression 741 = zcDeflate;
   const APassword: String = ''
); static;
```

Parameters

ZipFileName
File name of the ZIP file

Path
Path of directory to zip

Compression
Compression mode.

See also

- TZipFile Class 266

8.66.1.4 Create

Create a TZipFile

Unit

EZip 144

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
public
constructor Create;
```

See also

- TZipFile Class 266

8.66.1.5 Destroy

Unit

EZip 144

Syntax

Code (Delphi)

```
public
destructor Destroy;
   override;
```

Remarks

Destroy will close an open zipfile before disposing of it

See also

- TZipFile Class 266

8.66.1.6 Add

Add a file to the ZIP file

Add a memory file to the ZIP file

Add a memory file to the ZIP file

Add a memory file to the ZIP file. Allows programmer to specify the Local and Central Header data for more flexibility on what gets written. Minimal vailidation is done on the Header parameters; speficying bad options could result in a corrupted zip file.

Unit

EZip 144

Syntax

Code (Delphi)

```
public
procedure Add(
   const FileName: string;
   ArchiveFileName: string = '';
   Compression: TZipCompression 741 = zcDeflate
); overload;

procedure Add(
   Data: TBytes;
   ArchiveFileName: string;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   Compression: TZipCompression = zcDeflate
); overload;

procedure Add(
   Data: Classes.TStream;
   ArchiveFileName: string;
   Compression: TZipCompression = zcDeflate
); overload;

procedure Add(
   Data: Classes.TStream;
   LocalHeader: TZipHeader;
   CentralHeader: PZipHeader = nil
); overload;
```

Parameters

FileName
FileName to be added

ArchiveFileName
Path + Name of file in the arcive. If Ommitted, `ExtractFileName(FileName)` will be used.

Compression
Compression mode.

Data
Bytes to be added

ArchiveFileName
Path + Name of file in the arcive.

Compression
Compression mode.

Data
Stream of file to be added

ArchiveFileName
Path + Name of file in the arcive.

Compression
Compression mode.

Data
Stream of file to be added

LocalHeader
The local header data

CentralHeader
A Pointer to an optional central header. If no central Header is provided, the Local Header information is used.

See also

- TZipFile Class

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.66.1.7 Close

Unit
EZip 144

Syntax

Code (Delphi)

```
public
procedure Close;
```

Remarks
Closing is required to write the ZipFile's Central Directory to disk. Closing a file that is open for writing writes additonal metadata that is required for reading the file.

See also
- TZipFile Class 266

8.66.1.8 Extract

Extract a single file

Unit
EZip 144

Syntax

Code (Delphi)

```
public
procedure Extract(
   const FileName: string;
   const Path: string = '';
   CreateSubdirs: Boolean = True
); overload;
```

Parameters
FileName
File name in the archive

Path
Path to extract to disk

CreateSubdirs
The output should create sub directories specified in the ZIP file

Remarks
`FileName` specifies a file in the ZIP file. All slashes in ZIP file names should be '/'. The overload that takes an Integer may be useful when a ZIP file has duplicate filenames.

See also
- TZipFile Class 266

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.66.1.9 ExtractAll

Extract All files

Unit

EZip 144

Syntax

Code (Delphi)

```
public
procedure ExtractAll(
   const Path: string = ''
);
```

Parameters

Path

Path to extract to.

See also

- TZipFile Class 266

8.66.1.10 IndexOf

Translate from FileName to index in ZIP Central Header

Unit

EZip 144

Syntax

Code (Delphi)

```
public
function IndexOf(
   const FileName: string
): Integer;
```

Parameters

FileName

Path + Name of file in the arcive.

Return value

The index of the file in the archive, or -1 on failure.

Remarks

A ZIP file may have dupicate entries with the same name. This function will return the index of the first.

See also

- TZipFile Class 266

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

8.66.1.11 Open

Opens a ZIP file for reading or writing.

Unit
EZip 144

Syntax

Code (Delphi)

```
public
procedure Open(
   const ZipFileName: string;
   OpenMode: TZipMode
); overload;
```

Parameters
ZipFileName
Path to ZipFile

OpenMode
File Mode to open file. `zmWrite` Creates a new ZIP file for writing. `zmReadWrite` Opens the file for reading and allows adding additional new files. `zmRead` Opens the file for reading.

See also
- TZipFile Class 266

8.66.1.12 Read

Read a file from arcive to an array of Bytes

Get a stream to read a file from disk

Unit
EZip 144

Syntax

Code (Delphi)

```
public
procedure Read(
   const FileName: string;
   out Bytes: TBytes
); overload;

procedure Read(
   const FileName: string;
   out Stream: Classes.TStream;
   out LocalHeader: TZipHeader 747
); overload;
```

Parameters
FileName

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ZIP file FileName

Bytes
Output bytes

FileName
ZIP file FileName

Stream
Output Stream

LocalHeader
Local File header

Remarks

The overload that takes an Integer may be useful when a ZIP file has duplicate filenames.

The Stream returned by this function is a decomression stream wrapper around the interal Stream reading the zip file. You must Free this stream before using other TZipFile methods that change the contents of the ZipFile, such as Read or Add. The overload that takes an Integer may be useful when a ZIP file has duplicate filenames.

See also

- TZipFile Class 266

8.66.2 Properties

Comment 274	No description available
FileComment 275	No description available
FileCount 275	No description available
FileInfo 275	No description available
FileInfos 275	No description available
FileName 276	No description available
FileNames 276	No description available
Mode 276	No description available

See also

- TZipFile 266
- EZip 144

8.66.2.1 Comment

Unit

EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Comments can be a maximum of 65535 bytes long. If a longer comment is supplied, It is truncated before writing to the ZIP File.

See also

- TZipFile Class 266

8.66.2.2 FileComment

Unit

EZip 144

Remarks

File Comments can be changed for files opened in write mode at any point. The comment is written when the Central Directory is written to disk. Comments can be a maximum of 65535 bytes long. If a longer comment is supplied, It is truncated before writing to the ZIP File.

See also

- TZipFile Class 266

8.66.2.3 FileCount

Unit

EZip 144

Return value

Total files in ZIP File

See also

- TZipFile Class 266

8.66.2.4 FileInfo

Unit

EZip 144

Return value

TZipHeader of a File in the ZipFile

See also

- TZipFile Class 266

8.66.2.5 FileInfos

Unit

EZip 144

Return value

An array of the TZipHeader of the files in the ZIP file

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TZipFile Class 266

8.66.2.6 FileName

Unit

EZip 144

Return value

FileName of a File in the ZipFile

See also

- TZipFile Class 266

8.66.2.7 FileNames

Unit

EZip 144

Return value

An array of FileNames in the ZIP file

See also

- TZipFile Class 266

8.66.2.8 Mode

Unit

EZip 144

Return value

The mode the TZipFile is opened to

See also

- TZipFile Class 266

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part

9 All Interfaces

IEurekaLogger 278	Interface which holds a single entry in log file.
IEurekaLoggerParams 295	Sub-routine log entry.

9.1 IEurekaLogger

Interface which holds a single entry in log file.

Unit
ELogging 103

Syntax

Code (Delphi)

```
IEurekaLog(IUnknown)
['{76A72C59-97B1-4561-AE67-67F8B9F67EE8}'];
```

Remarks
This interface is returned by various ELog 360 functions. You can use it to alter properties of created log entry.

Methods 278	Methods

See also
- ELog Function 360
- ELogEnter Function 369

9.1.1 Methods

AddCheckPoint 280	Same as ELogAddCheckPoint 364 function.
AddResetSeparator 280	Same as ELogAddResetSeparator 364 function.
AddSeparator 280	Same as ELogAddSeparator 365 function.
Category 281	Changes category of the log entry.
Color 281	Changes color of the log entry.
Details 282	Changes details of the log entry.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Enter 283	Same as ELogEnter 369 function.
FontColor 283	Changes font color of the log entry.
Kind 284	Changes kind of the log entry.
Log 285	Same as ELog 360 function.
LogAssigned 287	Same as ELogAssigned 365 function.
LogAssignedPtr 287	Same as ELogAssignedPtr 366 function.
LogCurrency 288	Same as ELogCurrency 367 function.
LogDateTime 288	Same as ELogDateTime 368 function.
LogError 288	Same as ELogError 371 function.
LogException 289	Same as ELogException 371 function.
LogIf 289	Same as ELogIf 373 function.
LogIfDateTime 290	Same as ELogIfDateTime 374 function.
LogInterval 291	Same as ELogInterval 375 function.
LogMemoryAsHex 291	Same as ELogMemoryAsHex 376 function.
LogNote 291	Same as ELogNote 377 function.
LogPointer 292	Same as ELogPointer 379 function.
LogReminder 292	Same as ELogReminder 380 function.
LogWarning 293	Same as ELogWarning 381 function.
LogWindowHandle 293	Same as ELogWindowHandle 382 function.
LogWinError 294	Same as ELogWinError 383 function.
ResetCheckPoint 294	Same as ELogResetCheckPoint 380 function.

See also

- IEurekaLogger 278
- ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

9.1.1.1 AddCheckPoint

Same as ELogAddCheckPoint 364 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
procedure AddCheckPoint;
```

Remarks

This method duplicates ELogAddCheckPoint 364 function. Please, see ELogAddCheckPoint 364 function for more information.

See also

- ELogAddCheckPoint Function 364
- IEurekaLogger Interface 278

9.1.1.2 AddResetSeparator

Same as ELogAddResetSeparator 364 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
procedure AddResetSeparator;
```

Remarks

This method duplicates ELogAddResetSeparator 364 function. Please, see ELogAddResetSeparator 364 function for more information.

See also

- ELogAddResetSeparator Function 364
- IEurekaLogger Interface 278

9.1.1.3 AddSeparator

Same as ELogAddSeparator 365 function.

Unit

ELogging 103

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
public
procedure AddSeparator;
```

Remarks

This method duplicates ELogAddSeparator 365 function. Please, see ELogAddSeparator 365 function for more information.

See also

- ELogAddSeparator Function 365
- IEurekaLogger Interface 278

9.1.1.4 Category

Changes category of the log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function Category(
   const ACategoryName: String
): IEurekaLogger 278;
```

Parameters

ACategoryName

New category to be set. This is an arbitraty string.

Return value

A modified log entry with new category set up.

Remarks

This method changes category of the log entry. You may use it to group log entries into different categories. EurekaLog's categories start with 'EurekaLog'.

See also

- Color Method 281
- FontColor Method 283
- Kind Method 284
- Details Method 282
- IEurekaLogger Interface 278

9.1.1.5 Color

Changes color of the log entry.

Unit

ELogging 103

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
public
function Color(
   const AColor: DWORD
): IEurekaLogger 278;
```

Parameters

AColor

New color to be set.

Return value

A modified log entry with new color set up.

Remarks

This method changes color of the log entry. You may use it to paint log entries in different colors. Default is clWhite.

See also

- Category Method 281
- FontColor Method 283
- Kind Method 284
- Details Method 282
- IEurekaLogger Interface 278

9.1.1.6 Details

Changes details of the log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function Details(
   const ADetails: String
): IEurekaLogger 278;
```

Parameters

ADetails

New details to be set.

Return value

A modified log entry with new details set up.

Remarks

This method changes details of the log entry. Usually you do not need to call this.

See also

- Kind Method 284

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Category Method 281
- Color Method 281
- FontColor Method 283
- IEurekaLogger Interface 278

9.1.1.7 Enter

Same as ELogEnter 369 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function Enter(
   const AFuncName: String = ''
): IEurekaLoggerParams 295; overload;

function Enter(
   const ASelf: TObject;
   const AMethodName: String = ''
): IEurekaLoggerParams 295; overload;

function Enter(
   const ASelf: TClass;
   const AMethodName: String = ''
): IEurekaLoggerParams 295; overload;
```

Remarks

This method duplicates ELogEnter 369 function. Please, see ELogEnter 369 function for more information.

See also

- ELogEnter Function 369
- IEurekaLogger Interface 278

9.1.1.8 FontColor

Changes font color of the log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function FontColor(
   const AColor: DWORD
): IEurekaLogger 278;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AColor

New color to be set.

Return value

A modified log entry with new font color set up.

Remarks

This method changes font color of the log entry. You may use it to paint log entries in different colors. Default is clBlack.

See also

- Category Method 281
- Color Method 281
- Kind Method 284
- Details Method 282
- IEurekaLogger Interface 278

9.1.1.9 Kind

Changes kind of the log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function Kind(
   const AMsgType: TLogMsgType 691
): IEurekaLogger 278;
```

Parameters

AMsgType

New kind to be set.

Return value

A modified log entry with new kind set up.

Remarks

This method changes kind of the log entry. Usually you do not need to call this.

See also

- Details Method 282
- Category Method 281
- Color Method 281
- FontColor Method 283
- IEurekaLogger Interface 278

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

9.1.1.10 Log

Same as ELog 360 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function Log(
   const AMsg: String
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: String
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: Char
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: Integer
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: Word
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: LongWord;
   const AAsHex: Boolean = False
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: UInt64;
   const AAsHex: Boolean = False
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: Int64
): IEurekaLogger 278; overload;

function Log(
   const AMsg: String;
   const AValue: Extended;
   const ADigits: Integer = 2
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: Boolean
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TPoint
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TSmallPoint
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TRect
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TGUID
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TObject
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TExceptionPointers
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TExceptionRecord
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TContext
): IEurekaLogger[278]; overload;

function Log(
   const AMsg: String;
   const AValue: TResponse[713]
): IEurekaLogger[278]; overload;
```

Remarks

This method duplicates ELog[360] function. Please, see ELog[360] function for more information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ELog Function 360
- IEurekaLogger Interface 278

9.1.1.11 LogAssigned

Same as ELogAssigned 365 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogAssigned(
   const AMsg: String;
   const AValue: TObject
): IEurekaLogger 278; overload;
```

Remarks

This method duplicates ELogAssigned 365 function. Please, see ELogAssigned 365 function for more information.

See also

- ELogAssigned Function 365
- IEurekaLogger Interface 278

9.1.1.12 LogAssignedPtr

Same as ELogAssignedPtr 366 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogAssignedPtr(
   const AMsg: String;
   const AValue: Pointer
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogAssignedPtr 366 function. Please, see ELogAssignedPtr 366 function for more information.

See also

- ELogAssignedPtr Function 366
- IEurekaLogger Interface 278

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

9.1.1.13 LogCurrency

Same as ELogCurrency 367 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogCurrency(
   const AMsg: String;
   const AValue: Currency
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogCurrency 367 function. Please, see ELogCurrency 367 function for more information.

See also

- ELogCurrency Function 367
- IEurekaLogger Interface 278

9.1.1.14 LogDateTime

Same as ELogDateTime 368 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogDateTime(
   const AMsg: String;
   const AValue: TDateTime
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogDateTime function. Please, see ELogDateTime 368 function for more information.

See also

- ELogDateTime Function 368
- IEurekaLogger Interface 278

9.1.1.15 LogError

Same as ELogError 371 function.

Unit

ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
public
function LogError(
   const AMsg: String
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogError 371 function. Please, see ELogError 371 function for more information.

See also

- ELogError Function 371
- IEurekaLogger Interface 278

9.1.1.16 LogException

Same as ELogException 371 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogException(
   const E: TObject;
   const AMsg: String = ''
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogException 371 function. Please, see ELogException 371 function for more information.

See also

- ELogException Function 371
- IEurekaLogger Interface 278

9.1.1.17 LogIf

Same as ELogIf 373 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function LogIf(
   const AExpression: Boolean;
   const AMsg: String
): IEurekaLogger[278]; overload;

function LogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: String
): IEurekaLogger[278]; overload;

function LogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: Integer
): IEurekaLogger[278]; overload;

function LogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: TObject
): IEurekaLogger[278]; overload;
```

Remarks

This method duplicates ELogIf[373] function. Please, see ELogIf[373] function for more information.

See also

- ELogIf Function[373]
- IEurekaLogger Interface[278]

9.1.1.18 LogIfDateTime

Same as ELogIfDateTime[374] function.

Unit

ELogging[103]

Syntax

Code (Delphi)

```
public
function LogIfDateTime(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: TDateTime
): IEurekaLogger[278];
```

Remarks

This method duplicates ELogIfDateTime[374] function. Please, see ELogIfDateTime[374] function for more information.

See also

- ELogIfDateTime Function[374]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- IEurekaLogger Interface 278

9.1.1.19 LogInterval

Same as ELogInterval 375 function.

Unit
ELogging 103

Syntax

Code (Delphi)

```
public
procedure LogInterval(
   const AMsg: String;
   const AStart: Int64
);
```

Remarks
This method duplicates ELogInterval 375 function. Please, see ELogInterval 375 function for more information.

See also
- ELogInterval Function 375
- IEurekaLogger Interface 278

9.1.1.20 LogMemoryAsHex

Same as ELogMemoryAsHex 376 function.

Unit
ELogging 103

Syntax

Code (Delphi)

```
public
function LogMemoryAsHex(
   const AMsg: String;
   const ABuffer: Pointer;
   ASize: Integer
): IEurekaLogger 278;
```

Remarks
This method duplicates ELogMemoryAsHex 376 function. Please, see ELogMemoryAsHex 376 function for more information.

See also
- ELogMemoryAsHex Function 376
- IEurekaLogger Interface 278

9.1.1.21 LogNote

Same as ELogNote 377 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ELogging 103

Syntax

Code (Delphi)

```
public
function LogNote(
   const AMsg: String
): IEurekaLogger 278;
```

Remarks
This method duplicates ELogNote 377 function. Please, see ELogNote 377 function for more information.

See also
- ELogNote Function 377
- IEurekaLogger Interface 278

9.1.1.22 LogPointer

Same as ELogPointer 379 function.

Unit
ELogging 103

Syntax

Code (Delphi)

```
public
function LogPointer(
   const AMsg: String;
   const AValue: Pointer
): IEurekaLogger 278;
```

Remarks
This method duplicates ELogPointer 379 function. Please, see ELogPointer 379 function for more information.

See also
- ELogPointer Function 379
- IEurekaLogger Interface 278

9.1.1.23 LogReminder

Same as ELogReminder 380 function.

Unit
ELogging 103

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
public
function LogReminder(
   const AMsg: String
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogReminder 380 function. Please, see ELogReminder 380 function for more information.

See also

- ELogReminder Function 380
- IEurekaLogger Interface 278

9.1.1.24 LogWarning

Same as ELogWarning 381 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogWarning(
   const AMsg: String
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogWarning 381 function. Please, see ELogWarning 381 function for more information.

See also

- ELogWarning Function 381
- IEurekaLogger Interface 278

9.1.1.25 LogWindowHandle

Same as ELogWindowHandle 382 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogWindowHandle(
   const AMsg: String;
   const AValue: HWND
): IEurekaLogger 278;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This method duplicates ELogWindowHandle 382 function. Please, see ELogWindowHandle 382 function for more information.

See also

- ELogWindowHandle Function 382
- IEurekaLogger Interface 278

9.1.1.26 LogWinError

Same as ELogWinError 383 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function LogWinError(
   const AMsg: String;
   const ALastError: TSysErrorCode 737
): IEurekaLogger 278;
```

Remarks

This method duplicates ELogWinError 383 function. Please, see ELogWinError 383 function for more information.

See also

- ELogWinError Function 383
- IEurekaLogger Interface 278

9.1.1.27 ResetCheckPoint

Same as ELogResetCheckPoint 380 function.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
procedure ResetCheckPoint;
```

Remarks

This method duplicates ELogResetCheckPoint 380 function. Please, see ELogResetCheckPoint 380 function for more information.

See also

- ELogResetCheckPoint Function 380
- IEurekaLogger Interface 278

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

9.2 IEurekaLoggerParams

Sub-routine log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
IEurekaLoggerParams = interface(IEurekaLogger 278)
['{05A12A38-D3DA-49B6-8935-4427677AD740}'];
```

Remarks

A specialized version of IEurekaLogger 278 which allows you to specify additional parameters for current log entry.

Use P 298 method to log additional parameter.

Use D 297 or C 295 to indicate end of parameter list.

Use Ret 300 to indicate result of sub-routine.

Methods 295	Methods

9.2.1 Methods

C 295	Completes logging of function call (collapsed version).
D 297	Completes logging of function call.
P 298	Logs argument.
Ret 300	Logs return value.

See also

- IEurekaLoggerParams 295
- ELogging 103

9.2.1.1 C

Completes logging of function call (collapsed version).

Unit

ELogging 103

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
public
function C: IEurekaLoggerParams 295;
```

Return value

Returns log entry. This value could be used to log `Result`.

Remarks

Same as D 297 method, except function call will appear collapsed in the log.

Examples

Code (Delphi)

```
procedure DoSomething;
begin
   ELogEnter('DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething;
begin
   ELogEnter(Self, 'DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AValue: Integer);
begin
   ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AMsg: String; const AValue: Integer)
begin
   ELogEnter(Self, 'DoSomething').P('AMsg', AMsg).P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
function TObject.DoSomething(const AValue: Integer): Integer;
var
   L: IEurekaLoggerParams;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
begin
   L := ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code

   L.Ret(Result);
end;
```

See also

- D Method 297
- Ret Method 300
- IEurekaLoggerParams Interface 295

9.2.1.2 D

Completes logging of function call.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function D: IEurekaLoggerParams 295;
```

Return value

Returns log entry. This value could be used to log `Result`.

Remarks

Call this method to indicate that there are no more arguments to the function.

Important!

You have to call either D 297 or C 295 method on the returned IEurekaLoggerParams 295 interface. Otherwise nested log entries may have wrong identation in the resulting log file.

Examples

Code (Delphi)

```
procedure DoSomething;
begin
   ELogEnter('DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething;
begin
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   ELogEnter(Self, 'DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AValue: Integer);
begin
   ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AMsg: String; const AValue: Integer)
begin
   ELogEnter(Self, 'DoSomething').P('AMsg', AMsg).P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
function TObject.DoSomething(const AValue: Integer): Integer;
var
   L: IEurekaLoggerParams;
begin
   L := ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code

   L.Ret(Result);
end;
```

See also

- C Method 295
- Ret Method 300
- IEurekaLoggerParams Interface 295

9.2.1.3 P

Logs argument.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
function P(
   const AName, AValue: String
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: Int64
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: Pointer
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: TObject
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: Boolean
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: TGUID
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: TDateTime
): IEurekaLoggerParams; overload;

function P(
   const AName: String;
   const AValue: System.UInt64
): IEurekaLoggerParams; overload;
```

Parameters

AName

Name of the argument as it appears in source code.

AValue

Value of the argument.

Return value

Returns log entry with created new argument. This value could be used to add more arguments, to close function call, or to log `Result`.

Remarks

Use this method to log one or more arguments to sub-routine.

Important!

You have to call either D or C method on the returned IEurekaLoggerParams interface. Otherwise nested log entries may have wrong identation in the resulting log file.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
function Test(const AIndex: Integer; const ASpecifier: String): Integer;
var
   L: IEurekaLoggerParams;
begin
   L := ELogEnter('Test').
      P('AIndex', AIndex).
      P('ASpecifier', ASpecifier).
      D;

   // your code

   L.Ret(Result);
end;
```

See also

- D Method 297
- C Method 295
- Ret Method 300
- IEurekaLoggerParams Interface 295

9.2.1.4 Ret

Logs return value.

Unit

ELogging 103

Syntax

Code (Delphi)

```
public
procedure Ret(
   const AResult: String
); overload;

procedure Ret(
   const AResult: Integer
); overload;

procedure Ret(
   const AResult: Cardinal
); overload;

procedure Ret(
   const AResult: Int64
); overload;

procedure Ret(
   const AResult: System.UInt64
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure Ret(
   const AResult: TObject
); overload;

procedure Ret(
   const AResult: TResponse
); overload;

procedure Ret(
   const AResult: Boolean
); overload;

procedure Ret(
   const AResult: TDateTime
); overload;

procedure Ret;
   overload;
```

Parameters

AResult [in]

Value of Result variable.

Remarks

Use this method to log return value of the sub-routine.

Examples

Code (Delphi)

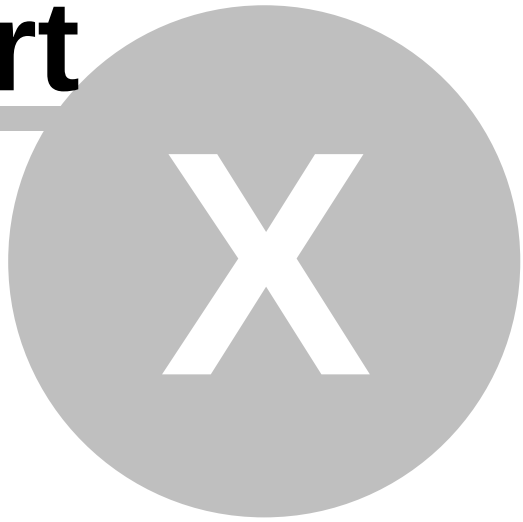
```
function Test(const AIndex: Integer; const ASpecifier: String): Integer;
var
   L: IEurekaLoggerParams;
begin
   L := ELogEnter('Test').
      P('AIndex', AIndex).
      P('ASpecifier', ASpecifier).
      D;

   // your code

   L.Ret(Result);
end;
```

See also

- D Method
- C Method
- P Method
- IEurekaLoggerParams Interface

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part



10 All Functions

AcquireThreadData 318	Returns thread information (any thread).
AddCustomDataToBugReport 319	Adds static data to bug report.
AddCustomWebField 320	Adds static data to web requests.
AllocatePageAlign 321	No description available
AllocationGranularity 322	No description available
AllocMemCountEx 322	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemCountMax 323	Represents the maximum number of allocated memory blocks.
AllocMemSizeEx 324	Represents the total size of allocated memory blocks.
AllocMemSizeMax 324	Represents the maximum size of allocated memory.
BeginThreadEx 325	Spawns a separate thread of execution.
BuildBugReport 328	Builds bug report from exception information.
CanCallMemWipeStack 328	Indicates if the `MemWipeStack` function should be called.
CheckELMagicCode 329	No description available
CheckHeap 330	Checks heap for consistency.
ConfigDelete 331	Deletes a key from the specified section.
ConfigFileName 331	Returns .ini-file name to store EurekaLog configuration.
ConfigIsRegistry 333	Indicates if EurekaLog configuration is stored in the registry or in .ini-file.
ConfigKeyPath 334	Returns registry key name to store EurekaLog configuration.
ConfigReadBool 335	Reads boolean value from specified key and section.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ConfigReadFloat 336	Reads float value from specified key and section.
ConfigReadInteger 337	Reads integer value from specified key and section.
ConfigReadString 338	Reads string value from specified key and section.
ConfigWriteBool 339	Writes boolean value to specified key and section.
ConfigWriteFloat 340	Writes float value to specified key and section.
ConfigWriteInteger 341	Writes integer value to specified key and section.
ConfigWriteString 342	Writes string value to specified key and section.
ConvertAddress 343	Convert address to code to real address of code.
ConvertRetAddr 344	Convert return address to instruction pointer.
CRC16Hash 345	Calculates CRC-16 checksum.
CRC32Hash 346	Calculates CRC-32 checksum.
CurrentEurekaLogOptions 8	No description available
CurrentInstance 348	Returns current instance.
DataProtect 349	Encrypts data.
DataUnprotect 350	Decrypts data.
DebugBreak 352	Causes a breakpoint exception to occur in the current process.
DecodeConfigValue 352	Low-level routine to decode data after reading from configuration storage.
DecodeInstruction 353	Decodes a single CPU instruction.
DisableMemLeaksRegistering 354	Disables memory leaks checks.
DisasmFunction 355	Disasms function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

DisasmInstruction 356	Disassebles a single CPU instruction.
DoneCheckFreeze 357	Stops background freeze detection thread.
DumpAllocationsToFile 358	Logs all allocated memory to file.
ELog 360	Creates a new log entry.
ELogAddCheckPoint 364	Logs a checkpoint.
ELogAddResetSeparator 364	Resets identation and adds a separator to the log.
ELogAddSeparator 365	Adds a separator to the log.
ELogAssigned 365	Checks if object is assigned and logs result.
ELogAssignedPtr 366	Checks if pointer is assigned and logs result.
ELogClear 367	Clears log file.
ELogCurrency 367	ELog for `Currency` type.
ELogDateTime 368	ELog for `TDateTime` type.
ELogEnter 369	Creates a new log entry about entering method/function/procedure.
ELogError 371	Adds an error to log file.
ELogException 371	ELog for `Exception` type.
ELogFlush 372	Flushes log to disk.
ELogIf 373	Adds a new log entry only if condition is true.
ELogIfDateTime 374	ELogIf for `TDateTime` type.
ELogInterval 375	Add a new time measurement to log file.
ELogMemoryAsHex 376	Adds a memory dump to log.
ELogNote 377	Adds a note to log file.
ELogOpen 378	Creates new log file.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ELogPointer 379	ELog for `Pointer` type.
ELogReminder 380	Adds a reminder to log file.
ELogResetCheckPoint 380	Resets checkpoint counter.
ELogWarning 381	Adds a warning to log file.
ELogWindowHandle 382	ELog for window handle.
ELogWinError 383	ELog for system error codes.
EnableMemLeaksRegistering 383	Enables memory leaks checks.
EncodeConfigValue 384	Low-level routine to encode data before saving to configuration storage.
EncryptBufferFree 385	Releases the encryption buffer.
EndThreadEx 386	Terminates the execution of a thread.
EnsureThreadStack 387	Sets the minimum size of the stack associated with the calling thread or fiber that will be available during any stack overflow exceptions. This is useful for handling stack overflow exceptions; the application can safely use the specified number of bytes during exception handling.
EnumSenders 387	Returns list of registered send method class names.
EurekaAllocMem 388	AllocMem allocates a memory block and initializes each byte to zero.
EurekaFreeMem 389	Frees a memory block previously allocated with EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.
EurekaGetMem 390	Allocates a memory block via EurekaLog memory filter.
EurekaReallocMem 391	Resizes a memory block.
EurekaRegisterExpectedMemoryLeak 392	Register an expected memory leak.
EurekaRegisterExpectedMemoryLeakDynArray 393	Register an expected memory leak (variant for dynamic arrays).
EurekaRegisterExpectedMemoryLeakStr 393	Register an expected memory leak

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	(variant for strings).
EurekaTryGetMem 394	Allocates a memory block via EurekaLog memory filter (w/o error checking).
EurekaUnregisterExpectedMemoryLeak 395	Removes memory block from excluded leaks.
EurekaUnregisterExpectedMemoryLeakDynArray 395	Removes memory block from excluded leaks (variant for dynamic arrays).
EurekaUnregisterExpectedMemoryLeakStr 396	Removes memory block from excluded leaks (variant for strings).
ExceptionManager 396	Exception manager.
ExceptionMessage 397	Retrieves error message for an exception.
ExpandEnvVars 398	Expands environment-variable strings.
FindDialogClass 399	Searches dialog class by name.
FindLowAddress 400	Tool function to obtain starting address to disassemble code block around known address.
FindScreenshotRect 400	Determinates screen area occupied by the specified object.
FindSender 401	Searches send method class by name.
FreezeNotifyStillAlive 401	Lets EurekaLog know that your main thread is still working.
GetActiveFormClass 402	Gets window class of active window.
GetActiveFormText 403	Gets caption of active window.
GetApplicationDescription 404	Gets application description.
GetApplicationName 404	Gets application name.
GetApplicationRect 405	Gets bounding rectangle for GUI application.
GetApplicationWnd 406	Returns application window (if applicable).
GetAsmSize 406	Returns size of a single machine opcode.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

GetCompanyName 407	Gets company name.
GetCompilationDate 408	Returns the compilation date of the specified executable module.
GetComputerName 409	Gets computer name user name.
GetControlClass 409	Gets class of active control.
GetControlText 410	Gets text of active control.
GetCurrentModuleCompilationDate 411	Gets compilation time of current module.
GetCurrentUserPrivileges 411	Gets description of user privileges.
GetDiskInfo 412	Gets total and free disk space size.
GetDisplayDPI 412	Gets screen DPI.
GetDisplayMode 413	Gets screen mode.
GetEnvVariable 413	Expands single environment variable.
GetEurekaLogLabel 414	Gets full EurekaLog version.
GetEurekaLogLicense 415	Gets EurekaLog license.
GetEurekaLogLicenseType 416	Gets EurekaLog license type.
GetEurekaLogPath 417	Obsolete function, do not use.
GetEurekaLogStatus 417	Gets EurekaLog stable/beta information.
GetEurekaLogVersion 418	Gets EurekaLog version.
GetExceptionModuleName 419	No description available
GetExceptionModuleVersion 419	No description available
GetFileIcon 420	No description available
GetFileSize 421	Gets file size.
GetFunctionSize 422	Returns size of the function in bytes.
GetLocationInfo 423	Returns human-readable information about code location.
GetLocationInfoStr 423	Returns human-readable information

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	about code location.
GetMainWnd 424	Returns main window of application (if applicable).
GetModuleType 425	Returns type of loaded executable module, based on analyzing its file header.
GetMXServerFromEmail 426	Retrieves mail server for e-mail address.
GetParentWnd 426	Returns active or top-most window, which can be used as a parent window for modal dialogs.
GetPerformanceCount 427	Returns timestamp.
GetReadableSize 428	Verifies that the caller has specified access to the specified range of memory.
GetTaskBarWnd 429	Returns window of application, which creates a taskbar button (if applicable).
GetTracer 430	Get stack tracing class by number.
GetTracerClass 431	Get stack tracing class by number.
GetUserEMail 432	No description available
GUIFont 432	Returns font for GUI applications.
HandleException 433	Handles exception.
HookWin32API 434	Adds a function to resource monitoring.
IfElse 435	A replacement for ternary `if` operator.
IniReadString 435	Reads string from .ini file.
InitCheckFreeze 436	Starts checking for freezes and hangs.
InitSalt 438	Generates random salt.
InitWER 440	Initializes EWER unit.
IniWriteString 440	Writes string to .ini file.
InstructionToString 441	Converts decoded CPU instruction to string represenstation.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IntoIDE 442	No description available
IsAppType 442	Checks if application is of specified type.
IsBPL 443	Checks if specified module is BPL file.
IsCBuilder 444	Indicates if current project was compiled by C++ Builder.
IsCGI 445	Checks if application is CGI application.
IsCGIConsole 445	Checks if application is CGI console application.
IsCGIWin 446	Checks if application is CGI Windows application.
IsCLX 446	Checks if application is CLX application.
IsCompiledWithPackages 447	Checks if this module is compiled with packages.
IsConsole 447	Checks if application is console application.
IsCPL 448	Checks if application is control panel application.
IsDebugged 449	Determines whether the calling process is being debugged by a user-mode debugger.
IsDLL 449	Checks if specified module is DLL file.
IsDynamicRTL 450	Indicates if current project was compiled with dynamic RTL DLL.
IsEurekaLogActive 3	Indicates if EurekaLog is active or disabled in this application.
IsEurekaLogActive 452	Indicates if EurekaLog is active or disabled in this application.
IsEurekaLogActiveInThread 5	Indicates if EurekaLog is active or disabled in the specified thread.
IsEurekaLogInstalled 2	Indicates if EurekaLog is installed in this application.
IsEXE 455	Checks if specified module is EXE file.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IsExpected 455	Checks if exception is an expected exception.
IsFMX 456	Checks if application is FMX application.
IsFreezeCheckEnabled 457	Indicate if freeze detection is currently active.
IsFreezeCheckInstalled 458	Indicate if freeze detection was initialized.
IsGUI 458	Checks if application is GUI application.
IsIntraWeb 459	Checks if application is IntraWeb application.
IsISAPI 459	Checks if application is ISAPI application.
IsLLVM 460	Indicates if current project is compiled by "new" C++ Builder compiler (e.g. LLVM).
IsMemLeaksEnabled 461	Indicate if memory leaks checks are enabled.
IsMemLeaksInstalled 461	No description available
IsModuleLibrary 462	Checks if specified module is DLL or BPL file.
IsReadableFile 463	Verifies that the caller has read access to the specified file.
IsRTLCompiledWithStackFrames 463	Determines whether the RTL was compiled with Stack Frames option enabled.
IsService 464	Checks if application is a win32 service application.
IsStackFramesProc 464	Determines whether the specified function has a stack frame.
IsValidBlockAddr 465	Verifies that the caller has read access to the specified range of memory.
IsValidClass 466	Verifies that the specified pointer is a valid class.
IsValidCodeAddr 467	Verifies that the caller has execute access to the specified range of memory.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IsValidHandle 468	Verifies that this is a valid kernel handle.
IsValidObject 468	Verifies that the specified pointer is a valid object.
IsVCL 469	Checks if application is VCL application.
IsVirtualized 470	Checks if application was instrumented/ virtualized.
IsWeb 470	Checks if application is web application.
IsWin10 471	Detect Windows 10 (minimum).
IsWin11 472	Detect Windows 11 (minimum).
IsWin64 472	Detect OS bitness.
IsWin7 473	Detect Windows 7 (minimum).
IsWin8 474	Detect Windows 8 (minimum).
IsWin81 474	Detect Windows 8.1 (minimum).
IsWinVista 475	Detect Windows Vista (minimum).
IsWinXP 476	Detect Windows XP (minimum).
IsWorkstation 476	Gets server/workstation status of current computer.
IsWritableFile 477	Verifies that the caller has write access to the specified file.
LastBugID 9	Returns BugID of last exception in current thread.
LocationToStr 478	Converts location information to single line string.
MainThreadTopOfStack 479	Returns top of the stack for the main thread.
MarkExpected 480	Marks exception as expected exception.
MD5Hash 480	Calculates MD5 hash.
MD5HMAC 481	Calculates HMAC-MD5.
MemLeaksGetName 482	No description available

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

MemLeaksName 483	Names a memory block.
MemLeaksNameDynArray 484	Names a memory block (variant for dynamic arrays).
MemLeaksNameStr 485	Names a memory block (variant for strings).
MemLeaksOwn 485	Sets ownership for memory block.
MemLeaksOwnDynArrayBoth 486	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayChild 487	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayParent 487	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnStr 488	Sets ownership for memory block (variant for strings).
MemLeaksSetParentBlock 489	No description available
MemLeaksSetParentBlockDynArrayBoth 489	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayChild 489	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayParent 490	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockStr 491	Sets parent-child relation between two memory blocks for grouping (variant for strings).
MemProtect 491	Encrypts process memory.
MemUnprotect 493	Decrypts process memory.
MemWipeStack 494	Clears unused stack of the current thread.
MonoFont 495	Returns fixed-width font.
NameThread 496	Names thread for debugging.
NameThreadForDebugging 496	Alias for `NameThread`, this overrides

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	System.Classes.TThread.NameThreadForDebugging.
PageSize 497	No description available
PauseFreezeCheck 497	Temporarily disables freeze checks.
PrimaryInstance 498	The same as MainInstance. See remarks for more info.
RaiseExpected 499	Raises expected exception.
RaiseFreezeException 499	Raises freeze/hang/deadlock exception for the specified thread.
RaiseOuterException 500	Raises an exception and captures the current one.
RaiseOuterExceptionExpected 501	Raises expected exception and captures the current one.
RegisterDialogClass 502	Registers dialog class.
RegisterDialogClassFirst 503	Registers dialog class.
RegisterEventAttachedFilesRequest 504	Registers OnAttachedFilesRequest event handler.
RegisterEventBeginReportGen 505	Registers OnBeginReportGen event handler.
RegisterEventCustomBugID 506	Registers OnCustomBugID event handler.
RegisterEventCustomButtonClick 507	Registers OnCustomButtonClick event handler.
RegisterEventCustomDataRequest 508	Registers OnCustomDataRequest event handler.
RegisterEventCustomFileName 509	Registers OnCustomFileName event handler.
RegisterEventCustomWebFieldsRequest 510	Registers OnCustomWebFieldRequest event handler.
RegisterEventEndReportGen 511	Registers OnEndReportGen event handler.
RegisterEventExceptionAction 512	Registers OnExceptionAction event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

RegisterEventExceptionError 513	Registers OnExceptionError event handler.
RegisterEventExceptionNotify 514	Registers OnExceptionNotify event handler.
RegisterEventHelpButtonClick 515	Registers OnHelpButtonClick event handler.
RegisterEventPasswordRequest 516	Registers OnPasswordRequest event handler.
RegisterEventRaise 517	Registers OnRaise event handler.
RegisterEventReproduceRequest 518	Registers OnReproduceRequest event handler.
RegisterEventZippedFilesRequest 519	Registers OnZippedFilesRequest event handler.
RegisterSender 520	Registers send method class.
RegisterSenderFirst 520	Registers send method class.
RegKeyCheck 521	Checks registry key for existing.
RegKeyRead 522	Reads string from registry key.
RegKeyWrite 523	Writes string to registry key.
ResumeFreezeCheck 524	Enables freeze checks back.
RSADecrypt 524	Decrypts encrypted data by using RSA cipher.
RSADestroyKey 527	Deletes RSA key.
RSAEncrypt 528	Encrypts "clear text" data by using RSA cipher.
RSAGenKey 531	Creates new RSA key.
RSALoadPrivateKey 532	Imports private key.
RSALoadPublicKey 533	Imports public key.
RSASavePrivateKey 534	Exports private key.
RSASavePublicKey 535	Exports public key.
RSASign 536	Signs data.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

RSAVerify 538	Verifies digital signature.
SafeExec 540	Executes routine with exception handling. Any exception is handled as fatal exception.
Screenshot 541	Takes screenshot.
ScreenshotFree 543	Release screenshot.
ScreenshotInit 543	Creates screenshot.
ScreenshotSave 544	Saves screenshot to file or stream.
SDBMHash 545	Calculates SDBM hash.
SecureFree 545	Wipes and releases memory.
SecureFree 547	No description available
SetAppType 547	Set a type of this application.
SetEurekaLogState 6	Enables or disables EurekaLog in this application.
SetEurekaLogState 549	Enables or disables EurekaLog in this application.
SetEurekaLogStateInThread 7	Enables or disables EurekaLog in the specified thread.
SetExceptionMessage 551	Sets or changes error message for an exception.
SetupConfig 552	Changes EurekaLog configuration storage.
SetUserEmail 553	No description available
SHA1Hash 553	Calculates SHA-1 hash.
SHA1HMAC 554	Calculates HMAC-SHA1.
SHA256Hash 555	Calculates SHA-2/SHA-256 hash.
SHA256HMAC 556	Calculates HMAC-SHA256.
ShowExpected 557	Shows expected exception.
SignatureGetInfo 558	Returns information about digital signature and its certificate of an

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	executable file.
TEADecrypt 559	Decrypts encrypted data by using TEA cipher.
TEADefaultKey 563	Returns pre-initialized default key for TEA.
TEADeriveKey 565	Initializes TEA key from password.
TEADeriveKeyHMAC 569	Initializes TEA key from password via MD5 HMAC.
TEADestroyKey 572	Fills TEA key with zeros.
TEAEncrypt 572	Encrypts "clear text" data by using TEA cipher.
TEAInitSessionKey 576	Generates random TEA key.
ThreadData 577	Returns thread information (current thread only).
ThreadDataEnum 578	Enumerates running threads.
TwofishDecrypt 578	Decrypts encrypted data by using Twofish cipher.
TwofishDeriveKey 580	Initializes Twofish key from password.
TwofishDeriveKeyHMAC 582	Initializes Twofish key from password using SHA-256 HMAC.
TwofishDestroyKey 583	Deletes Twofish key.
TwofishEncrypt 583	Encrypts "clear text" data by using Twofish cipher.
TwofishInitIV 585	Creates new IV.
TwofishInitKey 586	"Imports" (precalculates) Twofish key from RAW key value.
TZipCompressionToString 586	Converts ZIP compression method value to string representation.
UnRegisterEventAttachedFilesRequest 587	Unregisters previously registered OnAttachedFilesRequest event handler.
UnRegisterEventBeginReportGen 588	Unregisters previously registered OnBeginReportGen event handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

UnRegisterEventCustomBugID 589	Unregisters previously registered OnCustomBugID event handler.
UnRegisterEventCustomButtonClick 589	Unregisters previously registered OnCustomButtonClick event handler.
UnRegisterEventCustomDataRequest 590	Unregisters previously registered OnCustomDataRequest event handler.
UnRegisterEventCustomFileName 591	Unregisters previously registered OnCustomFileName event handler.
UnRegisterEventCustomWebFieldsRequest 592	Unregisters previously registered OnCustomWebFieldRequest event handler.
UnRegisterEventEndReportGen 593	Unregisters previously registered OnEndReportGen event handler.
UnRegisterEventExceptionAction 594	Unregisters previously registered OnExceptionAction event handler.
UnRegisterEventExceptionError 594	Unregisters previously registered OnExceptionError event handler.
UnRegisterEventExceptionNotify 595	Unregisters previously registered OnExceptionNotify event handler.
UnRegisterEventHelpButtonClick 596	Unregisters previously registered OnHelpButtonClick event handler.
UnRegisterEventPasswordRequest 597	Unregisters previously registered OnPasswordRequest event handler.
UnRegisterEventRaise 598	Unregisters previously registered OnRaise event handler.
UnRegisterEventReproduceRequest 598	Unregisters previously registered OnReproduceRequest event handler.
UnRegisterEventZippedFilesRequest 599	Unregisters previously registered OnZippedFilesRequest event handler.
WCTDone 600	Finalize EWCT unit.
WCTInit 600	Initialize EWCT unit.

10.1 AcquireThreadData

Returns thread information (any thread).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
function AcquireThreadData(
   const AThreadID: Cardinal;
   const AForce: Boolean = False
): TEurekaThreadData 219;
```

Parameters

AThreadID [in]

TID (Thread ID) of a thread to get information.

AForce [in, optional]

True: return dummy object for empty result, false (default): return nil for empty result.

Return value

Information about the AThreadID thread. You have to destroy the returned object. This value could be nil for rare cases (such as calling this function before/after EThreadsManager's initialization/finalization), or if AThreadID does not identify a valid running thread. The return value is always non-nil if AForce argument is True.

Remarks

Use this function to get information about any running thread.

You have to release the returned object. A new return value object is created for each function's call.

There is no need to use locking - because informational object 219 for the thread will be individual for each function's call. Any change for dynamic properties (such as Name, LastException, LastHTMLPage) will be ignored. Use ThreadData 577 function to modify thread's dynamic properties.

This function can return information about any running thread.

Note

This function returns information only about running threads. Return value will be empty if AThreadID identifies already terminated thread.

See also

- ThreadData Function 577
- ThreadDataEnum Function 578
- TEurekaThreadData Class 219

10.2 AddCustomDataToBugReport

Adds static data to bug report.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure AddCustomDataToBugReport(
   const AName, AValue: String
);
```

Parameters

AName [in]
Custom field name. Can be arbitrary text of your choice.

AValue [in, optional]
Custom field value. If this value is empty, custom field will be deleted/cleared. If this value is set, custom field will be assigned.

Remarks

This function can be used to add custom fields to all EurekaLog bug reports. The field's value is set when this function is called. Therefore, this function can only be used to set a static value.

Tip

Use the RegisterEventCustomDataRequest 508 function to register your custom code to supply a dynamic value for custom bug report fields.

See also

- RegisterEventCustomDataRequest function 508

10.3 AddCustomWebField

Adds static data to web requests.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure AddCustomWebField(
   const AName, AValue: String
);
```

Parameters

AName [in]
Custom web field name. Can be arbitrary text of your choice.

AValue [in, optional]
Custom web field value. If this value is empty, custom web field will be deleted/cleared. If this value is set, custom web field will be assigned.

Remarks

This function can be used to add custom fields to all EurekaLog web requests. The field's value is set when this

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

function is called. Therefore, this function can only be used to set a static value.

Tip
Use the RegisterEventCustomWebFieldsRequest 510 function to register your custom code to supply a dynamic value for custom web fields.

See also

- RegisterEventCustomWebFieldsRequest function 510

10.4 AllocatePageAlign

Allocates memory with alignment on current memory's page size boundary.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function AllocatePageAlign(
   const ASize: Cardinal;
   out AHolder: Pointer
): Pointer;
```

Parameters

ASize [in]
Size of memory block to allocate in bytes.

AHolder [out]
a pointer to backbone storage. This is the pointer which must be passed to FreeMem to release this memory block.

Return value

A pointer to memory block of `ASize` bytes aligned at page boundary.

Exceptions

System.SysUtils.EOutOfMemory	Not enought memory to perform request.

Remarks

This function is the same as AllocMem except it also aligns result on memory's page size boundary.

That's it, if memory's page size is 4 Kb - then result is always aligned on 4 Kb (i.e. have 12 zeros at low part of pointer when represented in binary).

Thus, this function have an overhead of maximum 2 size of memory's page. For example, if memory's page size is 4 Kb - then this function will allocate 12 Kb for `ASize` less than 4 Kb, 16 Kb - for `ASize` less than 8 Kb and so on.

You shouldn't pass returned pointer to any memory releasing routines (like FreeMem). Instead, call FreeMem

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

for `AHolder`.

Note
Page size may be different on different platforms. Do not make any assumptions about it. You can get current page size by calling PageSize 497.

See also

- AllocMem
- GetMem
- FreeMem
- PageSize 497
- AllocationGranularity 322

10.5 AllocationGranularity

Returns current memory's allocation granularity.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function AllocationGranularity: Cardinal;
inline;
```

Return value

Allocation granularity in bytes. Default is 64 Kb (65'536).

Remarks

The granularity for the starting address at which virtual memory can be allocated. For more information, see the VirtualAlloc function.

This is a `dwAllocationGranularity` field from the GetSystemInfo function.

See also

- PageSize 497
- AllocatePageAlign 321

10.6 AllocMemCountEx

Represents the total number of allocated memory blocks in a Delphi application.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function AllocMemCountEx: Int64;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Number of allocated memory blocks via EurekaLog's memory filter. Returns 0 if filter is not installed, no allocations were made, or all allocations were released.

Remarks

`AllocMemCountEx` is the number of currently allocated blocks of memory that were requested by the user. `AllocMemCountEx` is incremented each time a block of memory is allocated and is decremented each time a block of memory is freed. Use `AllocMemCountEx` to find out how many unfreed blocks of memory remain.

Tip
Unlike AllocMemCount, `AllocMemCountEx` is not deprecated and always returns actual information.

See also

- AllocMemCountMax Function 323
- AllocMemSizeEx Function 324
- IsMemLeaksInstalled Function 461
- IsMemLeaksEnabled Function 461
- EurekaGetMem Function 390

10.7 AllocMemCountMax

Represents the maximum number of allocated memory blocks.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function AllocMemCountMax: Int64;
```

Return value

Maximum of allocated memory blocks via EurekaLog's memory filter. Returns 0 if filter is not installed, or no allocations were made. This value is always equals or greater than AllocMemCountEx 322.

Remarks

`AllocMemCountMax` returns maximum of AllocMemCountEx 322. Use `AllocMemCountMax` to find maximum memory usage of your application.

Tip
Unlike AllocMemCount, `AllocMemCountMax` is not deprecated and always returns actual information.

See also

- AllocMemCountEx Function 322
- AllocMemSizeMax Function 324
- IsMemLeaksInstalled Function 461
- IsMemLeaksEnabled Function 461
- EurekaGetMem Function 390

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.8 AllocMemSizeEx

Represents the total size of allocated memory blocks.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function AllocMemSizeEx: Int64;
```

Return value

Size in bytes of all allocated memory blocks via EurekaLog's memory filter. Returns 0 if filter is not installed, no allocations were made, or all allocations were released.

Remarks

`AllocMemSizeEx` is the total size, in bytes, of all currently allocated blocks of memory in use by an application. Use `AllocMemSizeEx` to find out how many bytes of memory an application is currently using.

Tip

Unlike AllocMemSize, `AllocMemSizeEx` is not deprecated and always returns actual information.

Note

`AllocMemSizeEx` mimics AllocMemSize and indicate sum of all requested sizes - without additional overhead imposed by EurekaLog filter.

See also

- AllocMemSizeMax Function 324
- AllocMemCountEx Function 322
- IsMemLeaksInstalled Function 461
- IsMemLeaksEnabled Function 461
- EurekaGetMem Function 390

10.9 AllocMemSizeMax

Represents the maximum size of allocated memory.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function AllocMemSizeMax: Int64;
```

Return value

Size in bytes of maximum amount of all allocated memory blocks via EurekaLog's memory filter. Returns 0 if

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

filter is not installed, or no allocations were made. This value is always equals or greater than AllocMemSizeEx 324.

Remarks

`AllocMemSizeMax` returns maximum of AllocMemSizeEx 324. Use `AllocMemSizeMax` to find maximum memory usage of your application.

Tip
Unlike AllocMemSize, `AllocMemSizeMax` is not deprecated and always returns actual information.

See also

- AllocMemSizeEx Function 324
- AllocMemCountMax Function 323
- IsMemLeaksInstalled Function 461
- IsMemLeaksEnabled Function 461
- EurekaGetMem Function 390

10.10 BeginThreadEx

Spawns a separate thread of execution.

Unit

EBase 57

Syntax

Code (Delphi)

```
function BeginThreadEx(
   SecurityAttributes: Pointer;
   StackSize: LongWord;
   ThreadFunc: TThreadFunc;
   Parameter: Pointer;
   CreationFlags: LongWord;
   var ThreadId: TThreadID;
   const AThreadName: String = '';
   const AEnableEL: Boolean = Boolean(2
)): Integer;
```

Parameters

`ThreadAttributes` [in, opt]

A pointer to a `SECURITY_ATTRIBUTES` record that determines whether the returned handle can be inherited by child processes. If `ThreadAttributes` is nil, the handle cannot be inherited.

The lpSecurityDescriptor member of the `ThreadAttributes` record specifies a security descriptor for the new thread. If `ThreadAttributes` is nil, the thread gets a default security descriptor. The ACLs in the default security descriptor for a thread come from the primary token of the creator.

`StackSize` [in, opt]

The initial size of the stack, in bytes. The system rounds this value to the nearest page. If this parameter is zero, the new thread uses the default size for the executable. For more information, see Thread Stack Size.

`ThreadFunc` [in]

A pointer to the application-defined function to be executed by the thread. This pointer represents the

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

starting address of the thread.

Parameter [in, opt]
A pointer to a variable to be passed to the thread.

CreationFlags [in, opt]
Set of bits OR'ed together that are drawn from the following: CREATE_SUSPENDED: The thread is created in a suspended state, and does not run until the ResumeThread function is called. STACK_SIZE_PARAM_IS_A_RESERVATION: The StackSize parameter specifies the initial reserve size of the stack. If this flag is not specified, StackSize specifies the commit size.

ThreadId [out]
Receives the thread identifier (TID).

AThreadName [in, opt]
Defines thread's name. Thread name is arbitrary text string which can contain any combination of characters. Thread name will be visible in IDE's debugger. It also will be used by EurekaLog in bug reports. Default is empty name (no name; TID will be used to identify this thread).

AEnableEL [in, opt]
Defines EurekaLog status in created thread. This argument is ignored if you compile your application without EurekaLog. Default is the same state as current thread.

Return value

If the function succeeds, the return value is a handle to the new thread. The handle must be closed via CloseHandle.

If the function fails, the return value is 0. To get extended error information, call GetLastError.

Note
BeginThreadEx may succeed even if ThreadFunc points to data, code, or is not accessible. If the start address is invalid when the thread runs, an exception occurs, and the thread terminates. Thread termination due to a invalid start address is handled as an error exit for the thread's process. This behavior is similar to the asynchronous nature of CreateProcess, where the process is created even if it refers to invalid or missing dynamic-link libraries (DLLs).

Remarks

BeginThreadEx is EurekaLog's replacement for BeginThread function. BeginThreadEx offers two additional arguments, which allows you to control thread's name and EurekaLog's state in this thread.

EurekaLog will automatically handle any unhandled exception inside ThreadFunc (assuming that you did not set AEnableEL to False).

Use this routine or a TThreadEx 255 object to spawn separate threads of execution. BeginThreadEx spawns a new thread of execution and sets the global IsMultiThread variable, thereby making the heap thread-safe.

The number of threads a process can create is limited by the available virtual memory. By default, every thread has one megabyte of stack space. Therefore, you can create at most 2,048 threads. If you reduce the default stack size, you can create more threads. However, your application will have better performance if you create one thread per processor and build queues of requests for which the application maintains the context information. A thread would process all requests in a queue before processing requests in the next queue.

The new thread handle is created with the THREAD_ALL_ACCESS access right. If a security descriptor is not provided when the thread is created, a default security descriptor is constructed for the new thread using the primary token of the process that is creating the thread. When a caller attempts to access the thread with the OpenThread function, the effective token of the caller is evaluated against this security descriptor to grant or

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

deny access.

The newly created thread has full access rights to itself when calling the GetCurrentThread function.

If the thread is created in a runnable state (that is, if the CREATE_SUSPENDED flag is not used), the thread can start running before BeginThreadEx returns and, in particular, before the caller receives the handle and identifier of the created thread.

The thread execution begins at the function specified by the ThreadFunc parameter. If this function returns, the DWORD return value is used to terminate the thread in an implicit call to the ExitThread function. Use the GetExitCodeThread function to get the thread's return value.

The thread is created with a thread priority of THREAD_PRIORITY_NORMAL. Use the GetThreadPriority and SetThreadPriority functions to get and set the priority value of a thread.

When a thread terminates, the thread object attains a signaled state, satisfying any threads that were waiting on the object.

The thread object remains in the system until the thread has terminated and all handles to it have been closed through a call to CloseHandle.

Examples

Code (Delphi)

```
function ThreadFunc(Parameter: Pointer): Integer;
begin
   // No additional code needed inside thread func

   // - ... your code for the thread ...

   Result := 0; // to indicate "success"
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   TID: Cardinal;
begin
   // This code ignores any failures in thread, but
   // you may want to use GetExitCodeThread function.
   CloseHandle(BeginThreadEx(nil, 0, ThreadFunc, nil, 0, TID,
      // New argument: thread name
      'This is my thread with Parameter = ' +
         IntToHex(NativeUInt(nil), SizeOf(Pointer) * 2)));
end;
```

Threads launched with BeginThreadEx will be EurekaLog-enabled by default. You can supply optional Boolean argument for BeginThreadEx function to disable EurekaLog in thread, for example:

Code (Delphi)

```
BeginThreadEx(nil, 0, ThreadFunc, nil, 0, TID, 'Thread Name', False { di
```

See also

- EndThreadEx Function 386
- TThreadEx Class 255
- SetEurekaLogStateInThread Function

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- NameThread Function 496
- HandleException Function 433

10.11 BuildBugReport

Builds bug report from exception information.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
function BuildBugReport(
   const AExceptionInfo: TEurekaExceptionInfo 180;
   const ALogBuilderClass: TBaseLogBuilderClass = nil
): String;
```

Parameters

AExceptionInfo [in]
Information about exception to build bug report.

ALogBuilderClass [in, optional]
Specifies the class to build bug report. Default is `nil` - which means using the class specified in the LogBuilderClass 757 variable.

Return value

Textual representation of bug report. Exact value depends on used log builder class. Default log builder class is TLogBuilder 253. Builder class can be changed either via the `ALogBuilderClass` argument or via the LogBuilderClass 757 global variable.

Remarks

This function builds bug report for the specified exception information by using the default or the specified log builder class. This function can be used to re-create bug reports or create bug reports on demand.

Note

Please note that normally you don't have to call this function to get bug report for exceptions caught by EurekaLog as you can access exception's bug report at any time via TBaseDialog.BugReport property.

See also

- LogBuilderClass variable 757
- TBaseDialog.BugReport property

10.12 CanCallMemWipeStack

Indicates if the `MemWipeStack` function should be called.

Unit

ELowLevel 105

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
function CanCallMemWipeStack(
   const ASkipEurekaLogActivationCheck: Boolean = False
): Boolean;
```

Return value

`True` if the `MemWipeStack` function should be called, <\ `False` otherwise.

Remarks

Function returns `True` if the following conditions are met:

- The "Wipe Stack" option is enabled;
- Current thread has EurekaLog enabled;
- Current thread is not about to exhaust the stack space;
- Current thread is not in a "weird" state (for example: invalid `StackLimit` value);

Examples

Code (Delphi)

```
procedure TForm1.ClearStackOnIdle(Sender: TObject; var Done: Boolean);
begin
   if CanCallMemWipeStack then
      MemWipeStack;
   Done := True;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
   Application.OnIdle := Self.ClearStackOnIdle;
end;
```

See also

- MemWipeStack 494
- EnsureThreadStack 387

10.13 CheckELMagicCode

Checks double word for being one of the known EurekaLog's "magic codes" (signatures).

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function CheckELMagicCode(
   const ACode: DWord
): Boolean;
```

Parameters

ACode [in]

A value to check.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value
True if ACode is MagicNumber7, MagicNumber6 or MagicNumber5 constants. False - otherwise.

Remarks
"Magic code" is just a 4 bytes signature, which is used to indicate EurekaLog's data in executable.

Each version of EurekaLog uses its own magic code.

Magic code is also used to determinate correctness of password for encrypted debug information.

This routine is primary used in EDebugEL 66 unit, but also used in some other places where you need to check if this is valid EurekaLog's data or not.

10.14 CheckHeap

Checks heap for consistency.

Unit
EMemLeaks 109

Syntax

Code (Delphi)
```
procedure CheckHeap;
```

Remarks
This function will scan heap (dynamically allocated memory) and check each memory block for errors. The errors include overflow and underflow errors (i.e. overwriting header/footer of memory block).

This function will call MemLeaksError event handler for each found error. This usually results in showing bug report about the problem and terminating application.

Tip

The function will scan only memory blocks allocated via EurekaLog's memory filter 461 with enabled registering memory leaks 461.

Note

This function will do nothing if memory leaks checks are not enabled or if leOverflow value is in MemLeaksErrorsToIgnore global variable.

Warning

This function will block all memory operations during scanning. Also, the scan takes a lot of time to complete. Do not call this routine in production. This function is designed to be called for local debugging purposes only. I.e. call this routine to locate known memory corruption issue.

See also
- DumpAllocationsToFile Function 358

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.15 ConfigDelete

Deletes a key from the specified section.

Unit
EConfig 63

Syntax

Code (Delphi)
```
procedure ConfigDelete(
   const ASection, AName: String
);
```

Parameters
ASection
Section name to delete a key from. Can't be empty. Can't contain '\' or '/'.

AName
Key name to delete. Can't be empty.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks
This routine deletes a key from the section in EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Examples

Code (Delphi)
```
for X := FColumns.Count to OldCount do
begin
   ConfigDelete('Columns', 'Code_' + IntToStr(X));
   ConfigDelete('Columns', 'Name_' + IntToStr(X));
end;
```

See also
- EConfig 63
- RegKeyDelete
- IniWriteString 440

10.16 ConfigFileName

Returns .ini-file name to store EurekaLog configuration.

Unit
EConfig 63

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function ConfigFileName: String;
```

Return value

.ini-file name.

Exceptions

System.SysUtils.EAssertionFailed	If called when ConfigIsRegistry is True.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

Returns information about EurekaLog configuration storage location.

Returns .ini-file name to store EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 552 procedure.

Use this routine to implement custom ConfigReadXXX-like or ConfigWriteXXX-like routines.

Examples

Code (Delphi)

```
if ConfigIsRegistry then
begin
   ListView.RestoreFromRegistry(ConfigKeyPath + ListViewSubPath);
   BarManager.LoadFromRegistry(ConfigKeyPath + BarManagerSubPath);
end
else
begin
   ListView.RestoreFromIniFile(ConfigFileName);
   BarManager.LoadFromIniFile(ConfigFileName);
end;
```

See also

- SetupConfig 552
- ConfigIsRegistry 333
- ConfigKeyPath 334

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.17 ConfigIsRegistry

Indicates if EurekaLog configuration is stored in the registry or in .ini-file.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigIsRegistry: Boolean;
```

Return value

True - EurekaLog configuration is stored in the registry. Use ConfigKeyPath 334 to get registry key name.

False - EurekaLog configuration is stored in the .ini-file. Use ConfigFileName 331 to get .ini-file name.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

Returns information about EurekaLog configuration storage location.

Indicates if EurekaLog configuration is stored in the registry or in .ini-file.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 552 procedure.

Use this routine to implement custom ConfigReadXXX-like or ConfigWriteXXX-like routines.

Examples

Code (Delphi)

```
if ConfigIsRegistry then
begin
   ListView.RestoreFromRegistry(ConfigKeyPath + ListViewSubPath);
   BarManager.LoadFromRegistry(ConfigKeyPath + BarManagerSubPath);
end
else
begin
   ListView.RestoreFromIniFile(ConfigFileName);
   BarManager.LoadFromIniFile(ConfigFileName);
end;
```

See also

- SetupConfig 552
- ConfigKeyPath 334

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ConfigFileName 331

10.18 ConfigKeyPath

Returns registry key name to store EurekaLog configuration.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigKeyPath: String;
```

Return value

Registry key name with leading and trailing '\' delimiters.

Exceptions

System.SysUtils.EAssertionFailed	If called when ConfigIsRegistry is False.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

Returns information about EurekaLog configuration storage location.

Returns registry key name to store EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 552 procedure.

Use this routine to implement custom ConfigReadXXX-like or ConfigWriteXXX-like routines.

Examples

Code (Delphi)

```
if ConfigIsRegistry then
begin
   ListView.RestoreFromRegistry(ConfigKeyPath + ListViewSubPath);
   BarManager.LoadFromRegistry(ConfigKeyPath + BarManagerSubPath);
end
else
begin
   ListView.RestoreFromIniFile(ConfigFileName);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   BarManager.LoadFromIniFile(ConfigFileName);
end;
```

See also

- SetupConfig 552
- ConfigIsRegistry 333
- ConfigFileName 331

10.19 ConfigReadBool

Reads boolean value from specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigReadBool(
   const ASection, AName: String;
   const ADefault: Boolean = False
): Boolean;
```

Parameters

ASection

Section name. Can't be empty. Can't contain '\' or '/'.

AName

Key name. Can't be empty.

ADefault

Default value, if key or section doesn't exits or key's value have a wrong format. Optional.

Return value

Returns key's value on success or `ADefault` in case of non-existent key or section or wrong value's format.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Reads boolean value from EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 338 function.

Examples

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
CheckUpdates.Checked := ConfigReadBool(IDESectionName, IDECanCheckUpdate
```

See also

- ConfigReadString 338
- ConfigReadInteger 337
- ConfigReadFloat 336

10.20 ConfigReadFloat

Reads float value from specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigReadFloat(
   const ASection, AName: String;
   const ADefault: Extended = 0
): Extended;
```

Parameters

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

ADefault
Default value, if key or section doesn't exits or key's value have a wrong format. Optional.

Return value

Returns key's value on success or `ADefault` in case of non-existent key or section or wrong value's format.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Reads float value from EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 338 function.

Examples

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
StartDate := ConfigReadFloat(Name, Key);
```

See also

- ConfigReadString 338
- ConfigReadInteger 337
- ConfigReadBool 335

10.21 ConfigReadInteger

Reads integer value from specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigReadInteger(
   const ASection, AName: String;
   const ADefault: Int64 = 0
): Int64;
```

Parameters

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

ADefault
Default value, if key or section doesn't exits or key's value have a wrong format. Optional.

Return value

Returns key's value on success or `ADefault` in case of non-existent key or section or wrong value's format.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Reads integer value from EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 338 function.

Examples

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
Width := ConfigReadInteger(IniCoordinatesSection, IniWidthParam, 745);
Height := ConfigReadInteger(IniCoordinatesSection, IniHeightParam, 468);
Left := ConfigReadInteger(IniCoordinatesSection, IniLeftParam, ScreenRec
Top := ConfigReadInteger(IniCoordinatesSection, IniTopParam, ScreenRect.
FWindowState := ConfigReadInteger(IniCoordinatesSection, IniWindowsState
```

See also

- ConfigReadString 338
- ConfigReadBool 335
- ConfigReadFloat 336

10.22 ConfigReadString

Reads string value from specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
function ConfigReadString(
   const ASection, AName: String;
   const ADefault: String = ''
): String;
```

Parameters

ASection

Section name. Can't be empty. Can't contain '\' or '/'.

AName

Key name. Can't be empty.

ADefault

Default value, if key or section doesn't exits. Optional.

Return value

Returns key's value on success or `ADefault` in case of non-existent key or section.

This value can hold arbitrary bytes, you don't need to decode it in any way.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Reads string value from EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
function GetUserEmail: String;
begin
   Result := ConfigReadString(IniUserSettingsSection, IniUserSettingsEMa
end;
```

See also

- ConfigWriteString 342
- ConfigReadInteger 337
- ConfigReadBool 335
- ConfigReadFloat 336
- ConfigDelete 331
- SetupConfig 552
- RegKeyRead 522
- IniReadString 435
- DecodeConfigValue 352

10.23 ConfigWriteBool

Writes boolean value to specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure ConfigWriteBool(
   const ASection, AName: String;
   const AValue: Boolean
);
```

Parameters

ASection

Section name. Can't be empty. Can't contain '\' or '/'.

AName

Key name. Can't be empty.

AValue

Value to write. Mandatory. If you need to delete value - use ConfigDelete 331 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes boolean value to EurekaLog configuration.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 342 function.

Examples

Code (Delphi)

```
ConfigWriteBool(IDESectionName, IDECanCheckUpdatesName, CheckUpdates.Che
```

See also

- ConfigWriteString 342
- ConfigWriteInteger 341
- ConfigWriteFloat 340
- ConfigDelete 331

10.24 ConfigWriteFloat

Writes float value to specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure ConfigWriteFloat(
   const ASection, AName: String;
   const AValue: Extended
);
```

Parameters

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

AValue
Value to write. Mandatory. If you need to delete value - use ConfigDelete 331 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes float value to EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 342 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
if StartDate = 0 then
begin
   StartDate := Now;
   ConfigWriteFloat(Name, Key, StartDate);
end;
```

See also

- ConfigWriteString 342
- ConfigWriteInteger 341
- ConfigWriteBool 339
- ConfigDelete 331

10.25 ConfigWriteInteger

Writes integer value to specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure ConfigWriteInteger(
   const ASection, AName: String;
   const AValue: Int64
);
```

Parameters

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

AValue
Value to write. Mandatory. If you need to delete value - use ConfigDelete 331 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes integer value to EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 342 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
for n := 0 to 7 do
begin
   ConfigWriteInteger(IniCoordinatesSection, 'ColPro' + IntToStr(n), //
      Integer(SendMessage(GetDlgItem(ActivePage, ID_LISTMEMO),
      LVM_GETCOLUMNWIDTH, n, 0)));
end;
```

See also

- ConfigWriteString 342
- ConfigWriteBool 339
- ConfigWriteFloat 340
- ConfigDelete 331

10.26 ConfigWriteString

Writes string value to specified key and section.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure ConfigWriteString(
   const ASection, AName: String;
   const AValue: String
);
```

Parameters

ASection

Section name. Can't be empty. Can't contain '\' or '/'.

AName

Key name. Can't be empty.

AValue

Value to write. Mandatory, but can be an empty string. If you need to delete value - use ConfigDelete 331 procedure instead. Writing empty string will not delete value.

Do not encode this value. It can hold arbitrary bytes and they all will be stored properly.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Writes string value to EurekaLog configuration.

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Examples

Code (Delphi)

```
procedure SetUserEmail(const AUserEmail: String);
begin
   ConfigWriteString(IniUserSettingsSection, IniUserSettingsEMail, AUser
end;
```

See also

- ConfigReadString 338
- ConfigWriteInteger 341
- ConfigWriteBool 339
- ConfigWriteFloat 340
- ConfigDelete 331
- SetupConfig 552
- RegKeyWrite 523
- IniWriteString 440
- EncodeConfigValue 384

10.27 ConvertAddress

Convert address to code to real address of code.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function ConvertAddress(
   const AAddr: Pointer
): Pointer;
```

Parameters

AAddr [in]

Address to code. Usually points to routine.

Return value

Adjusted address to code. See remarks. Can be the same value as `AAddr`.

Remarks

This routine is used for hooks.

If you have an address to routine - it can points to code itself or to JMP instruction to code. Typical example is import table.

This function takes address and checks if there is a JMP instruction at this address.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- If there is no JMP - then function returns value unchanged.
- If there is JMP opcode - then function returns target of that JMP.

Function doesn't raise access violation in case of unavailable memory.

Examples

Code (Delphi)

```
var
   P: Pointer;
begin
   P := @GetWindowText; // P can point to routine itself or stub with JM
   P := ConvertAddr(P); // now P points to routine itself
end;
```

See also

- ConvertRetAddr 344
- IsValidBlockAddr 465
- IsValidCodeAddr 467

10.28 ConvertRetAddr

Convert return address to instruction pointer.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function ConvertRetAddr(
   const AReturnAddr: Pointer
): Pointer;
```

Parameters

AReturnAddr [in]

Return address. It is usually obtained by analyzing call stack or via `ReturnAddress`.

Return value

Pointer to CALL opcode. This value is always less than `AReturnAddr`.

Remarks

This routine is used for hooks.

If you have a return address and want to show/indicate caller - then you can use this function to convert return address to instruction pointer in caller's source. This function is needed because return address points to the next line in source file.

Function doesn't raise access violation in case of unavailable memory.

Warning

Function does **NOT** check if the `AReturnAddr` is a valid return address. Do not pass arbitrary pointers,

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

pass return addresses only.

Note

This is a wrapper for `EDisAsm.GetAddrByRetAddr` function.

Examples

Code (Delphi)

```
var
   P: Pointer;
begin
   P := ReturnAddress; // P is a return address to caller (e.g. next lir
   P := ConvertRetAddr(P); // Now P points to the exact caller of this c
end;
```

See also

- ConvertAddress 343
- GetAddrByRetAddr
- GetTargetAddrByCallAddr

10.29 CRC16Hash

Calculates CRC-16 checksum.

Unit

EHash 97

Syntax

Code (Delphi)

```
function CRC16Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TCRC16Hash 621; overload;

function CRC16Hash(
   const AData: RawByteString
): TCRC16Hash 621; overload;
```

Parameters

`AData` [in]
A data to calculate checksum.

`ADataSize` [in]
Size of `AData` in bytes.

Return value

CRC-16 checksum of the specified `AData`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. CRCs are based on the theory of cyclic error-correcting codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

Important!

Unlike cryptographic hash functions (like MD5 or SHA1), CRC is an easily reversible function, which makes it unsuitable for use in digital signatures.

Numerous varieties of cyclic redundancy checks have been incorporated into technical standards. `CRC16Hash` function uses CRC-16-CCITT algorithm. Polynomial is `$1021`. Starting value is `$FFFF`.

Note

This routine is used to calculate BugID from string source.

See also

- CRC32Hash 346

10.30 CRC32Hash

Calculates CRC-32 checksum.

Unit

EHash 97

Syntax

Code (Delphi)

```
function CRC32Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TCRC32Hash 622; overload;

function CRC32Hash(
   const AData: RawByteString
): TCRC32Hash 622; overload;
```

Parameters

`AData` [in]

A data to calculate checksum.

`ADataSize` [in]

Size of `AData` in bytes.

Return value

CRC-32 checksum of the specified `AData`.

Remarks

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. CRCs are based on the theory of cyclic error-correcting

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

Important!

Unlike cryptographic hash functions (like MD5 or SHA1), CRC is an easily reversible function, which makes it unsuitable for use in digital signatures.

Numerous varieties of cyclic redundancy checks have been incorporated into technical standards. `CRC32Hash` function uses CRC-32-IEEE 802.3 algorithm. Polynomial is `$EDB88320`. Starting value is `$FFFFFFFF`. For example, this implementation matches PHP's implementation of CRC-32.

See also

- CRC16Hash 345

10.31 CurrentEurekaLogOptions

Returns current EurekaLog's options.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function CurrentEurekaLogOptions: TEurekaModuleOptions 216;
```

Return value

Current EurekaLog's options.

Remarks

This function returns options for EurekaLog in current executable module. Options are present if EurekaLog was installed.

Important!

If EurekaLog is not active - any attempt to call this function will result in error message "*Cannot use ''CurrentEurekaLogOptions'' function in module "XYZ" without activate EurekaLog*" and immediate process termination.

This behaviour is "as designed". It's done to avoid cases when you ship your application with EurekaLog's code but without EurekaLog active.

Default options are set based on EurekaLog's projects settings - as set by you in IDE expert or standalone Settings Editor Tool. Options can be changed at run-time at any time.

Note

Some options are read-only. That's because these options can not be changed after activation.

`CurrentEurekaLogOptions` represents global EurekaLog's options. These options can also be captured locally. In this case changes to global options will have no effect for captured options.

For example, when exception is raised - it creates a copy of options. Thus, if you change options globally - this

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

will have no effect on already active exceptions.

If you want to change options for such exceptions - you should use local copy of options (i.e. `Options` property for dialogs, `Options` argument in event handlers, etc.).

See TEurekaModuleOptions 216 for detailed description of available EurekaLog's options.

Note
It's highly recommended to use enviroment variables when possible instead of customizing options at run-time. That's because some exceptions can occur before you get chance to setup options.

Note
This is a wrapper for EModules.CurrentEurekaLogOptions function.

Examples

Code (Delphi)

```
// Disable sending if run on my machine
if ApplicationRunOnDeveloperMachine then
   CurrentEurekaLogOptions.SenderClasses := '';
```

10.32 CurrentInstance

Returns current instance.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function CurrentInstance: HINST;
```

Return value

Current instance. See remarks.

Remarks

This function returns current instance (i.e. module handle).

It's used as replacement for HInstance variable.

"Current instance" means instance with EurekaLog which corresponds to caller of this function. Usually this is HInstance of the module, since often caller and EurekaLog's resides in the same executable module.

However, it can be a different value in case of packaged application (EurekaLog's code may reside in package which is not the same module as caller).

Important!
Do not use HInstance when working with EurekaLog. Use `CurrentInstance` instead.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
Options := EurekaLogOptions(CurrentInstance);
```

See also

- PrimaryInstance 498
- HInstance
- MainInstance

10.33 DataProtect

Encrypts data.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure DataProtect(
   const AData: Pointer;
   const ADataSize: Cardinal;
   out AProtectedData: Pointer;
   out AProtectedDataSize: Cardinal;
   const ASameMachine: Boolean = False
); overload;

function DataProtect(
   const AData: RawByteString;
   const ASameMachine: Boolean = False
): RawByteString; overload;

procedure DataProtect(
   var AData: TEncryptBuffer 671;
   const ASameMachine: Boolean = False
); overload;

procedure DataProtect(
   const AData: TEncryptBuffer 671;
   out AProtectedData: TEncryptBuffer 671;
   const ASameMachine: Boolean = False
); overload;
```

Parameters

AData [in]
Sensitive data to encrypt.

ADataSize [in]
Size of AData.

AProtectedData [out]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Encrypted data. Can be deleted by FreeMem function (but we recommend to use SecureFree 545 function instead).

AProtectedDataSize [out]
Size of AProtectedData.

ASameMachine [in, optional]
False (default) - encrypt data using currently logged in user; True - encrypt data using computer info only.

AData [in]
Sensitive data to encrypt.

AData [in, out]
On input: sensitive data to encrypt. **Must** be allocated with AllocMem function.
On output: encrypted data (size will be different). Can be deleted by EncryptBufferFree 385 or SecureFree 545 functions.

AProtectedData [out]
Encrypted data (size will be different). Delete this data by EncryptBufferFree 385 or SecureFree 545 functions.

Return value

Encrypted data (size will be different).

Remarks

This function encrypts data. Typically, only a user with the same logon credential as the user who encrypted the data can decrypt the data (when ASameMachine is False). In addition, the encryption and decryption usually must be done on the same computer.

You can use this function to save data that you want to decrypt later. If you want to protect sensitive data in your process memory while running - use MemProtect 491 function.

Call the DataUnprotect 350 function to decrypt data encrypted with the DataProtect function. When you have finished using the sensitive information, clear it from memory by calling the FillChar function.

The function creates a session key to perform the encryption. The session key is derived again when the data is to be decrypted.

The function also adds a Message Authentication Code (MAC) (keyed integrity check) to the encrypted data to guard against data tampering.

See also

- DataUnprotect Function 350
- MemProtect Function 491

10.34 DataUnprotect

Decrypts data.

Unit

EEncrypt 80

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure DataUnprotect(
   const AProtectedData: Pointer;
   const AProtectedDataSize: Cardinal;
   out AData: Pointer;
   out ADataSize: Cardinal
); overload;

function DataUnprotect(
   const AProtectedData: RawByteString
): RawByteString; overload;

procedure DataUnprotect(
   var AProtectedData: TEncryptBuffer 671
); overload;

procedure DataUnprotect(
   const AProtectedData: TEncryptBuffer 671;
   out AData: TEncryptBuffer 671
); overload;
```

Parameters

AProtectedData [in]
Encrypted data to decrypt.

AProtectedDataSize [in]
Size of AProtectedData.

AData [out]
Decrypted sensitive data. Can be deleted by FreeMem function (but we recommend to use SecureFree 545 function instead).

ADataSize [out]
Size of AData.

AProtectedData [in]
Encrypted data to decrypt.

AProtectedData [in, out]
On input: encrypted data to decrypt. **Must** be allocated with AllocMem function.
On output: unencrypted sensitive data (size will be different). Can be deleted by EncryptBufferFree 385 or SecureFree 545 functions.

AData [out]
Unencrypted sensitive data (size will be different). Delete this data by EncryptBufferFree 385 or SecureFree 545 functions.

Return value

Unencrypted sensitive data (size will be different).

Remarks

DataUnprotect function decrypts memory that was encrypted using the DataProtect 349 function.

You must encrypt and decrypt the data using the same user account and same computer.

When you have finished using the sensitive information, clear it from memory by calling the FillChar function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- DataProtect Function 349

10.35 DebugBreak

Causes a breakpoint exception to occur in the current process.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
procedure DebugBreak;
```

Remarks

This routine is used to break into debugger. This allows the calling thread to signal the debugger - for example to handle the exception.

If you are running your project under debugger and the debugger stops inside this function - this means that EurekaLog is notifying debugger about a problem (possibly about a fatal one). You can the check call stack to understand what this problem is, and possibly investigate the issue before default handling/processing will take place. You can also continue the execution to let EurekaLog handle the problem. Naturally, this "break into debugger" behaviour does not happen outside of the debugger.

Warning

This function raises exception! If there is no debugger attached - then the exception will go into your exception handlers, which can cause unwanted error dialogs or even termination of the process. Always check if debugger is present before calling this function.

Examples

Code (Delphi)

```
if IsDebugged then
DebugBreak;
```

See also

- IsDebugged 449

10.36 DecodeConfigValue

Low-level routine to decode data after reading from configuration storage.

Unit

EConfig 63

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function DecodeConfigValue(
   const AValue: String
): String;
```

Parameters

AValue

Encoded data, which was retrieved from configuration storage. This data should be processed by EncodeConfigValue 384 before saving to storage.

Return value

Original raw data.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

Decodes data which was previosly encoded by EncodeConfigValue 384.

Usually you don't need to use this function, because you're encouraged to use ConfigReadString 338 or it's wrapper. These ConfigReadXYZ functions perform such decoding automatically.

See also

- EncodeConfigValue 384
- ConfigReadString 338

10.37 DecodeInstruction

Decodes a single CPU instruction.

Unit

EDisAsm 78

Syntax

Code (Delphi)

```
function DecodeInstruction(
   Address: PAddress;
   const AExternalProcess: Boolean = False
): TInstruction 687;
```

Parameters

Address [in]

A pointer to CPU instruction to decode.

Return value

Decoded CPU instruction at `Address`. Exact structure of `TInstruction` depends on current CPU. Please, refer to source code of the unit for your CPU (such as EDisAsmX8632.pas, EDisAsmX8664.pas, etc.).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This is low-level disassembler routine which decodes a single CPU instruction and stores detailed information into `TInstruction` record. The decoded information includes prefixes, opcode, MOD/SIB bytes, displacement and immediate values - plus evaluated information: such as pointer to next instruction, flags, sizes of various data, etc. Exact structure of `TInstruction` depends on current CPU. Please, refer to source code of the unit for your CPU (such as EDisAsmX8632.pas, EDisAsmX8664.pas, etc.).

The decoded information can be used "as is" or it can be converted into textual representation by InstructionToString 441 function.

Note
You can use DisasmInstruction 356 function if you need to disassemble a single CPU instruction into textual representation and you don't need decoded information (`TInstruction` record).

See also

- DisasmInstruction function 356
- InstructionToString function 441
- DisasmFunction function 355

10.38 DisableMemLeaksRegistering

Disables memory leaks checks.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure DisableMemLeaksRegistering;
```

Remarks

This function temporally disables regestering memory leaks. This is a global state, which can be checked with IsMemLeaksEnabled function 461. You can enable regestering memory leaks back with EnableMemLeaksRegistering function 383.

- When registering memory leaks is enabled, EurekaLog will collect call stack information of each allocated memory block. Additional information (such as name, grouping, etc.) will also be stored.
- When registering memory leaks is disabled, EurekaLog will not collect call stack and additional information. Overhead will be significantly reduced.

See also description of EMemLeaks unit 109.

Important!
This function will not disable generating report about already registered memory leak. To disable memory leak report - use ActivateMemoryLeakCheck function.

Note
You can use EurekaRegisterExpectedMemoryLeak function 392 to register an expected memory leak or to disable leak reporting for already allocated memory block.

Caution!

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

`DisableMemLeaksRegistering`/`EnableMemLeaksRegistering` can not be called recursively. If you make two calls to `DisableMemLeaksRegistering` followed to one call to `EnableMemLeaksRegistering` - then registering of memory leaks will be enabled.

See also

- EnableMemLeaksRegistering Function 383
- IsMemLeaksEnabled Function 461
- ActivateMemoryLeakCheck Function
- EurekaRegisterExpectedMemoryLeak Function 392

10.39 DisasmFunction

Disasms function.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
function DisasmFunction(
   const Address: Pointer;
   const AExceptionAddress: Pointer = nil;
   const ASize: Integer = 0;
   const ALines: Integer = 0;
   const AOptions: TEurekaModuleOptions 216 = nil;
   const AEncrypt: Boolean = False
): String;
```

Parameters

`Address` [in]
A pointer to beginning of the function. Size of the function is determinated from `AExceptionAddress`, `ASize`, `ALines` or virtual region side if neither param is specified.

`AExceptionAddress` [in, optional]
A pointer to exception address inside function. If this argument is specified then the function will mark corresponding line in disassembly output as "< Exception Here". Function size will also be increased to include this address (if necessary).

`ASize` [in, optional]
Function's size in bytes. If this argument is set then the function will stop disassebmling only after specified number of bytes. Otherwise it will stop at first RET instruction or if `ALines`, `AExceptionAddress` or end of virtual region is reached.

`ALines` [in, optional]
Maximum number of lines (i.e. instructions) to disassemble after `AExceptionAddress`. Have no effect if `AExceptionAddress` is not specified.

Return value

A textual representation of disassembled function including comments, marks, addresses, RAW bytes and decoded instructions.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function is used to generate disasm view in bug reports. This function uses EDisAsm 78 unit.

Note
When neither of `AExceptionAddress`, `ASize` and `ALines` arguments are specified: this function will disassemble machine code to the end of the allocated memory region or until RET instruction is encountered.

This function can be used to disassemble arbitrary block of machine code. `Address` does not have to point to beginning of some function. It can point to any CPU instruction. One can use FindLowAddress 400 to obtain starting address from known `AExceptionAddress` value. Return value of `FindLowAddress` can be used as `Address` param.

Note
When this function is used with FindLowAddress 400, it's recommended that FindLowAddress's `ALines` argument equals to DisasmFunction's `ALines` argument.

See also

- FindLowAddress function 400
- DisasmInstruction function 356
- DisasmInstruction function 422

10.40 DisasmInstruction

Disassebles a single CPU instruction.

Unit

EDisAsm 78

Syntax

Code (Delphi)

```
function DisasmInstruction(
   var Address: PAddress;
   const AEncrypt: Boolean = False
): String;
```

Parameters

`Address` [in, out]

A pointer to CPU instruction. On input: a pointer to CPU instruction to disassemble. On output: a pointer to the next CPU instruction.

Return value

Textual representation of disassemled CPU instruction. This includes `Address`, RAW bytes, instruction itself and optional comments.

Remarks

Function disassembles a single CPU instruction, returns its textual representation and advances `Address` to the next instruction. The last can be used to walk through code while diassembling it.

Sample output:

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code

```
006989A2 E805C2F8FF CALL -$073DFB ; ($00624BB0) Unit1.TForm1.CheckVisibl
```

- First part ("006989A2" in the above example) repeats `Address` and it's always 8 hex characters for 32-bit process and 16 hex characters for 64-bit process.
- Second part ("E805C2F8FF" in the above example) indicate RAW instruction bytes. Its size varies from 2 characters (1 byte) up to 30 characters (15 bytes) for x86-32 - depending on actual CPU instruction. These bytes includes any prefix, opcode, MOD/RM byte, SIB byte, displacement bytes, immediate value bytes.
- Third part ("CALL -$073DFB" in the above example) shows actual textual representation of disassembled instruction.
- Fourth part ("; ($00624BB0) Unit1.TForm1.CheckVisible" in the above example; optional) contains an optional comment for the instruction (such as call/jmp destination or string data dump).

Note

This hi-level function is a simple combination of low-level DecodeInstruction 353 and InstructionToString 441. This function is used as base for DisasmFunction 355 function.

See also

- DecodeInstruction function 353
- InstructionToString function 441
- DisasmFunction function 355

10.41 DoneCheckFreeze

Stops background freeze detection thread.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
procedure DoneCheckFreeze;
```

Remarks

This function will stop running background freeze detection thread launched by InitCheckFreeze 436. This function will do nothing if no such thread is running.

Use this function to completely stop freeze detection.

Important!

This function should be called from main thread.

Tip

Use PauseFreezeCheck 497 to temporarily turn off freeze detection - without full stop/re-initialization.

See also

- InitCheckFreeze Function 436
- IsFreezeCheckEnabled Function 457

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- PauseFreezeCheck Function 497

10.42 DumpAllocationsToFile

Logs all allocated memory to file.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure DumpAllocationsToFile(
   const AFileName: String;
   const ACount: Boolean = True;
   const AGroup: Boolean = True;
   const AGroupHeuristic: Boolean = True;
   const ASort: Boolean = True;
   const AFilter: TDumpFilterFunc 623 = nil;
   const ACompare: TDumpCompareFunc 622 = nil;
   const ACompareForCount: TDumpCompareFunc 622 = nil
);
```

Parameters

AFileName [in]
A fully qualified valid file name. File's folder must exists and be writable. File will be (re)created.

ACount [in, optional]
`True` - group duplicate memory blocks, `False` - report each memory block individually.

AGroup [in, optional]
`True` - group child blocks with parent, `False` - report all memory blocks.

AGroupHeuristic [in, optional]
`True` - group child blocks with parent objects by using call stack heuristic, `False` - group only explicitly parented blocks. This argument is ignored when `AGroup` is `False`.

ASort [in, optional]
`True` - sort memory blocks, default sort is: blocks with most count (i.e. same type and call stack) will be first, then blocks of bigger size, same size is sorted by type: first - objects, then dyn-arrays/strings, then RAW data; `False` - do not sort memory blocks, list blocks in order of allocation (older - first, newer - last).

AFilter [in, optional]
A custom filter for filtering out memory blocks.

ACompare [in, optional]
A custom comparator for sorting.

ACompareForCount [in, optional]
A custom comparator for counting. Positive and negative results are considered to be the same (e.g. "not equal"). (This argument could be the same as `ACompare`)

Remarks

This function will log all allocated memory blocks to the specified file. Each memory block will be logged with

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

call stack, properties (i.e. name, type, size, count) and short memory dump. Additionally a total memory statistics will be written.

This is a diagnostic function which can be used to find reasons for high memory usage, collect memory usage statitics, find hidden memory leaks, etc.

Counting

`DumpAllocationsToFile` will count memory blocks. Memory blocks are considered to be the same if these blocks have the same size, same type, same name and same call stack (up to 6 lines). Same memory block will be grouped together - only first block will be logged (with its call stack, dump and properties). Other blocks will not be logged. Instead, a count will be added to a header of the first logged memory blocks.

You may override a default comparison method by specifying your own `ACompareForCount`.

Typically, you are interested in memory blocks with higher counts. For example, when memory block is allocated 400 times - it may be an indication that such memory blocks are not released when needed.

Parenting

`DumpAllocationsToFile` may group child leaks with parent leak. Any memory block can be declared as child memory block to another memory block - you can do this via MemLeaksSetParentBlock function 489 or MemLeaksOwn function 485. Additionally, any memory allocated withing constructor of any object will be declared as child memory to this object - this is determinated by a heuristic check using call stacks.

Grouping by parent allows you to view a lot of memory blocks as a single object. This feature is used to simplify report: it filters out non-important entries, showing only root parent for all sub-children. The idea behind this: if parent is leaked - then all children are leaked as well. However, you are not interested in children leak, because there is no bug. The bug is leaked parent. Thus, we hide leaked children, show leaked parent, now you can easily identificate root cause for bug. You solve bug - no more parent leak, thus no more children leak. Of course, if parent is not leaking, but some child are leaking - a child will be included in report.

Filtering

If you get many unrelated memory blocks included in your report, then you can supply a custom filter function (`AFilter`) to remove any particular memory block from report. Since run-time address change on each application's launch, you can not use exact address to filter out memory blocks. You have to rely on block's size, type, call stack, name, or, perhaps, an allocation order.

Sorting

Not all memory blocks are equally interesting. For examples, RAW data are often allocated from objects. Thus, objects are usually more interesting than RAW data.

Tip
You can use parenting (see above) to eliminate such cases.

You can use sorting to spot interest blocks and place them first. Default sorting prefers repeated blocks (with higher count), then it prefers large blocks (by grouped total size, not size of single blocks), then comes block's type (object, array, string or RAW), named/unnamed.

You can also supply a custom comparator function (`ACompare`) to define your own preferred sort order.

Naming

Usually it is difficult to identificate memory blocks inside memory report. While there is a lot of information (call stack, type, size, dump) - this is not enough for many cases. For example, if memory is allocated in cycle (like "while" cycle to read a document and create entries for each document's block). Such memory blocks will look the same in report, it is not possible to distinguish between them.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

You can use MemLeaksName 483 or MemLeaksOwn 485 functions to mark any memory block with arbitrary name tag. Name tag will be displayed in header for memory block in the report.

Tip
EurekaLog is capable of extracting class name from objects and "Name" property from components. Use custom names for other types of memory blocks.

Sizing

Headers for memory blocks indicate two sizes: single and total. Single is a size of allocation for each memory block inside the entry. Total is size occupied by entire entry. For duplicate (repeated) blocks: total size = single size * count. For parent-child grouping: parent total size = single parent size + each(total child size).

Tip
The function will log only memory blocks allocated via EurekaLog's memory filter 461 with enabled registering memory leaks 461.

Warning
This function will block all memory operations during logging. Also, the logging takes a lot of time to complete. Do not call this routine in production. This function is designed to be called for local debugging purposes only. I.e. call this routine to locate known memory leak issue.

Warning
While your application may work after calling this function, it is best to terminate the process immediatelly after calling this function. For technical reasons heap memory may not be in valid state after calling this function. Thus, you may encounter memory-related crashes.

See also

- CheckHeap Function 330
- TDumpFilterFunc Type 623
- TDumpCompareFunc Type 622

10.43 ELog

Creates a new log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
function ELog: IEurekaLogger;
   overload;

procedure ELog(
   const AMsg: String
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure ELog(
   const AMsg: String;
   const AValue: String
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: PChar
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Char
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Integer
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Word
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: LongWord;
   const AAsHex: Boolean = False
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: UInt64;
   const AAsHex: Boolean = False
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Int64
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Extended;
   const ADigits: Integer = 2
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: Boolean
); overload;

procedure ELog(
   const AMsg: String;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const AValue: TPoint
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TSmallPoint
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TRect
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TGUID
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TObject
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TExceptionPointers
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TExceptionRecord
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TContext
); overload;

procedure ELog(
   const AMsg: String;
   const AValue: TResponse
); overload;
```

Parameters

AMsg [in]
A message to log.

AValue [in]
A value to log.

AAsHex [in]
Indicate is value should be logged in hex format (like '0045FDA0'). Default is false.

ADigits [in]
Indicate how many digits should be listed after decimal separator (e.g. 2 means '0.00'). Default is 2.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Returns interface to manupulate new log entry.

Remarks

This function adds a new log entry. Use this function to add entries to log. You may alter created log entry (for example, change category, color, etc.) via methods of returned IEurekaLogger 278 interface.

There are several overloads: The overload without arguments will not log anything immediately - instead it returns IEurekaLogger logger to add customized entries. The overload with a single `AMsg` argument will log text "as is". The overload with `AMsg` and `AValue` will log text as "AMsg = AValue".

Examples

Code (Delphi)

```
ELog('Log started');
ELog('SomeVar', SomeVar);
```

Code (Delphi)

```
ELog.Category('My Category').Log('Log started');
ELog.Category('My Category').Log('SomeVar', SomeVar);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).Log('Log started');
ELog.Category('My Category').FontColor(clRed).Log('SomeVar', SomeVar);
```

See also

- ELogEnter Function 369
- ELogNote Procedure 377
- ELogError Procedure 371
- ELogWarning Procedure 381
- ELogReminder Procedure 380
- ELogAssignedPtr Function 366
- ELogAssigned Function 365
- ELogCurrency Procedure 367
- ELogDateTime Procedure 368
- ELogWinError Procedure 383
- ELogWindowHandle Procedure 382
- ELogPointer Procedure 379
- ELogMemoryAsHex Procedure 376
- ELogException Procedure 371
- ELogIf Procedure 373
- ELogIfDateTime Procedure 374
- ELogAddCheckPoint Procedure 364
- ELogResetCheckPoint Procedure 380
- ELogAddSeparator Procedure 365
- ELogAddResetSeparator Procedure 364
- ELogInterval Procedure 375

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.44 ELogAddCheckPoint

Logs a checkpoint.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogAddCheckPoint;
```

Remarks
Increments checkpoint counter and add new checkpoint entry to the log.

Examples

Code (Delphi)
```
ELogResetCheckPoint;

// do something dangerous
ELogAddCheckPoint;

// do something dangerous
ELogAddCheckPoint;

// do something dangerous
ELogAddCheckPoint;
```

See also
- ELogResetCheckPoint Function 380

10.45 ELogAddResetSeparator

Resets identation and adds a separator to the log.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogAddResetSeparator;
```

Remarks
Same as ELogAddSeparator 365, except it additionally resets all current identation back to zero.

Note

Calling this function inside any of logged function will broke a flow of called functions, because each

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

logged call increases identation.

See also
- ELogAddSeparator Function 365

10.46 ELogAddSeparator

Adds a separator to the log.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogAddSeparator;
```

Remarks
This function adds a simple separation line to visually indicate separation between log blocks.

See also
- ELogAddResetSeparator Function 364

10.47 ELogAssigned

Checks if object is assigned and logs result.

Unit
ELogging 103

Syntax

Code (Delphi)
```
function ELogAssigned(
   const AMsg: String;
   const AValue: TObject
): Boolean;
```

Parameters
AMsg [in]
A name of the object.

AValue [in]
An object to log.

Return value
Returns `True` is `AValue` is assigned, `False` otherwise.

Remarks
This functions adds an object status to the log. A value of `NIL` will be logged as "not assigned", everything

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

else will be logged as "assigned".

Examples

Code (Delphi)

```
if not ELogAssigned('SomeObject', SomeObject) then
Exit;
```

See also

- ELog Function 360
- ELogAssignedPtr Function 366

10.48 ELogAssignedPtr

Checks if pointer is assigned and logs result.

Unit

ELogging 103

Syntax

Code (Delphi)

```
function ELogAssignedPtr(
   const AMsg: String;
   const AValue: Pointer
): Boolean;
```

Parameters

AMsg [in]
A name of the pointer.

AValue [in]
A pointer to log.

Return value

Returns `True` is `AValue` is assigned, `False` otherwise.

Remarks

This functions adds a pointer status to the log. A value of `NIL` will be logged as "not assigned", everything else will be logged as "assigned".

Examples

Code (Delphi)

```
if not ELogAssignedPtr('SomePtrVar', SomePtrVar) then
Exit;
```

See also

- ELog Function 360
- ELogPointer Function 379

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- ELogMemoryAsHex Function 376
- ELogAssigned Function 365

10.49 ELogClear

Clears log file.

Unit
ELogging 103

Syntax

Code (Delphi)

```
procedure ELogClear;
```

Remarks
Erases all content of the log file. Use this function to clear log of old entries.

See also
- ELogOpen Function 378
- ELog Function 360

10.50 ELogCurrency

ELog for `Currency` type.

Unit
ELogging 103

Syntax

Code (Delphi)

```
procedure ELogCurrency(
   const AMsg: String;
   const AValue: Currency
);
```

Parameters
AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks
Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
ELogCurrency('CurrVar', CurrVar);
```

Code (Delphi)

```
ELog.Category('My Category').LogCurrency('CurrVar', CurrVar);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogCurrency('CurrVar', Cur
```

See also

- ELog Function 360

10.51 ELogDateTime

ELog for `TDateTime` type.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogDateTime(
   const AMsg: String;
   const AValue: TDateTime
);
```

Parameters

AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks

Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)

```
ELogDateTime('DateVar', DateVar);
```

Code (Delphi)

```
ELog.Category('My Category').LogDateTime('DateVar', DateVar);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogDateTime('DateVar', Dat
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ELog Function 360

10.52 ELogEnter

Creates a new log entry about entering method/function/procedure.

Unit

ELogging 103

Syntax

Code (Delphi)

```
function ELogEnter(
   const AFuncName: String = ''
): IEurekaLoggerParams 295; overload;

function ELogEnter(
   const ASelf: TObject;
   const AMethodName: String = ''
): IEurekaLoggerParams 295; overload;

function ELogEnter(
   const ASelf: TClass;
   const AMethodName: String = ''
): IEurekaLoggerParams 295; overload;
```

Parameters

AFuncName
Name of the logged function or procedure.

ASelf
Object or class which method is called.

AMethodName
Name of the logged method. Must belong to `ASelf`.

Return value

Returns created log entry.

Remarks

Use this function to add entries to log when entering sub-routine. The call to this function must be the first call in the sub-routine.

EurekaLog will log exit from sub-routine automatically. However, if you want to manually log early exit or specify a result - use Ret 300 method.

You may alter created log entry (for example, change category, color, etc.) via methods of IEurekaLogger 278 interface.

Important!

You have to call either D 297 or C 295 method on the returned IEurekaLoggerParams 295 interface. Otherwise

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

nested log entries may have wrong identation in the resulting log file.

Examples

Code (Delphi)

```
procedure DoSomething;
begin
   ELogEnter('DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething;
begin
   ELogEnter(Self, 'DoSomething').D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AValue: Integer);
begin
   ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
procedure TObject.DoSomething(const AMsg: String; const AValue: Integer)
begin
   ELogEnter(Self, 'DoSomething').P('AMsg', AMsg).P('AValue', AValue).D;

   // your code
end;
```

Code (Delphi)

```
function TObject.DoSomething(const AValue: Integer): Integer;
var
   L: IEurekaLoggerParams;
begin
   L := ELogEnter(Self, 'DoSomething').P('AValue', AValue).D;

   // your code

   L.Ret(Result);
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- ELogEnter Function 369
- IEurekaLoggerParams Interface 295

10.53 ELogError

Adds an error to log file.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogError(
   const AMsg: String
);
```

Parameters

AMsg [in]
An error text.

Remarks

Same as ELog 360, but adds an error instead of a normal text message.

Examples

Code (Delphi)

```
ELogError('Something is NOT OK');
```

Code (Delphi)

```
ELog.Category('My Category').LogError('Something is NOT OK');
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogError('Something is NOT
```

See also

- ELog Function 360
- ELogNote Function 377
- ELogWarning Function 381
- ELogReminder Function 380

10.54 ELogException

ELog for `Exception` type.

Unit

ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure ELogException(
   const E: TObject;
   const AMsg: String = ''
);
```

Parameters

E [in]
An exception to log.

AMsg [in, optional]
A name of exception.

Remarks

Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)

```
try
   ...
except
   on E: Exception do
   begin
      ELogException(E);
      HandleException(E);
   end;
end;
```

Code (Delphi)

```
try
   ...
except
   on E: Exception do
   begin
      ELogException(E, 'Exception in SomeFunc');
      HandleException(E);
   end;
end;
```

See also

- ELog Function 360
- ELogAssignedPtr Function 366
- ELogMemoryAsHex Function 376

10.55 ELogFlush

Flushes log to disk.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogFlush(
   const ACloseAllFunctions: Boolean = False
);
```

Remarks
This function will write all cached log entries to a file on the disk. Call this function before terminating application unexpectedly. This function will do nothing if log caching is not enabled;

See also
- ELogClear Function 367
- ELogOpen Function 378

10.56 ELogIf

Adds a new log entry only if condition is true.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogIf(
   const AExpression: Boolean;
   const AMsg: String
); overload;

procedure ELogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: String
); overload;

procedure ELogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: Integer
); overload;

procedure ELogIf(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: TObject
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AExpression [in]
An expression to check.

AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks

`AExpression` is `False`: the function does nothing. `AExpression` is `True`: the function acts as ELog 360.

Examples

Code (Delphi)

```
ELogIf(Var <> 0, 'Var', Var);
```

Code (Delphi)

```
ELog.Category('My Category').LogIf(Var <> 0, 'Var', Var);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogIf(Var <> 0, 'Var', Var
```

See also

- ELog Function 360
- ELogIfDateTime Function 374

10.57 ELogIfDateTime

ELogIf for `TDateTime` type.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogIfDateTime(
   const AExpression: Boolean;
   const AMsg: String;
   const AValue: TDateTime
);
```

Parameters

AExpression [in]
An expression to check.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks
Same as ELogIf 373. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)
```
ELogIfDateTime(DateVar <> 0, 'DateVar', DateVar);
```

Code (Delphi)
```
ELog.Category('My Category').LogIfDateTime(DateVar <> 0, 'DateVar', Date
```

Code (Delphi)
```
ELog.Category('My Category').FontColor(clRed).LogIfDateTime(DateVar <> 0
```

See also
- ELogIf Function 373
- ELog Function 360

10.58 ELogInterval

Add a new time measurement to log file.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogInterval(
   const AMsg: String;
   const AStart: Int64
);
```

Parameters
AMsg [in]
A message to log.

AStart [in]
A return value of GetPerformanceCount 427 function.

Remarks
This function will act as ELog 360 where `AValue` is replaced with time interval, starting at `AStart` and ending at current time.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
Start := GetPerformanceCount;
// your code
ELogInterval('Code took the following time', Start);
```

See also

- GetPerformanceCount Function 427
- ELog Function 360

10.59 ELogMemoryAsHex

Adds a memory dump to log.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogMemoryAsHex(
   const AMsg: String;
   const ABuffer: Pointer;
   ASize: Integer
);
```

Parameters

AMsg [in]
A message to log.

ABuffer [in]
A value to log.

ASize [in]
A size of ABuffer in bytes.

Remarks

Dumps memory from ABuffer to log file.

Examples

Code (Delphi)

```
ELogMemoryAsHex('Stream', Stream.Memory, Stream.Size);
```

Code (Delphi)

```
ELog.Category('My Category').LogMemoryAsHex('Stream', Stream.Memory, Str
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogMemoryAsHex('Stream', S
```

See also

- ELogPointer Function 379
- ELogAssignedPtr Function 366
- ELog Function 360

10.60 ELogNote

Adds a note to log file.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogNote(
   const AMsg: String
);
```

Parameters

AMsg [in]
A note text.

Remarks

Same as ELog 360, but adds a note instead of a normal text message.

Examples

Code (Delphi)

```
ELogNote('Everything is OK');
```

Code (Delphi)

```
ELog.Category('My Category').LogNote('Everything is OK');
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogNote('Everything is OK'
```

See also

- ELog Function 360
- ELogError Function 371
- ELogWarning Function 381
- ELogReminder Function 380

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.61 ELogOpen

Creates new log file.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogOpen(
   const AFileName: String = '';
   const ABufferSize: Cardinal = 1024*1024
);
```

Parameters

AFileName [in, opt]
An absolute full file name for the new log file. We recommend to use `.csl` file extension.

ABufferSize [in, opt]
Size of the buffer in bytes. Default is 1 Mb. Set to 0 to disable buffering.

Remarks

This function creates new log file. Add subsequent calls to ELog 360-like functions will write to `AFileName`.

A call to this function should be the first call in your application (e.g. before you call any of other logging routines).

If non-zero `ABufferSize` is specified: logging will write into a buffer. Buffer will be flushed to disk from time to time, or you can flush the buffer to disk at any time by calling the ELogFlush 372 function.

If `ABufferSize` is 0: buffering will be disabled, logging will write directly to the file on disk. Application's performance may be sighnificantly reduced.

Note

If you call any of ELog 360 function before calling this function - log entries will be added to internal log. Internal log will be copied to real log file once you call `ELogOpen`.

If you do not want old entries created earlier - call ELogClear 367 before calling `ELogOpen`.

Note

A call to this function may be ignored, if application is already streaming into log file, which is owned by other module (exe/DLL) in application.

Examples

Code (Delphi)

```
ExeName := ParamStr(0);
LogFileName := ChangeFileExt(ExtractFileName(ExeName), '.csl');
LogFolder := ExpandEnvVars('%APPDATA%'); // ESysInfo unit
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
FullLogFileName := IncludeTrailingPathDelimiter(LogFolder) + LogFileName

ELogOpen(FullLogFileName);
```

See also
- ELogFlush Function 372
- ELogClear Function 367
- ELog Function 360

10.62 ELogPointer

ELog for `Pointer` type.

Unit
ELogging 103

Syntax

Code (Delphi)

```
procedure ELogPointer(
   const AMsg: String;
   const AValue: Pointer
);
```

Parameters
AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks
Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)

```
ELogPointer('PointerVar', PointerVar);
```

Code (Delphi)

```
ELog.Category('My Category').LogPointer('PointerVar', PointerVar);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogPointer('PointerVar', P
```

See also
- ELog Function 360
- ELogAssignedPtr Function 366
- ELogMemoryAsHex Function 376

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.63 ELogReminder

Adds a reminder to log file.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogReminder(
   const AMsg: String
);
```

Parameters

AMsg [in]
A reminder text.

Remarks

Same as ELog 360, but adds a reminder instead of a normal text message.

Examples

Code (Delphi)

```
ELogReminder('TODO: refactor this code');
```

Code (Delphi)

```
ELog.Category('My Category').LogReminder('TODO: refactor this code');
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogReminder('TODO: refactc
```

See also

- ELog Function 360
- ELogNote Function 377
- ELogError Function 371
- ELogWarning Function 381

10.64 ELogResetCheckPoint

Resets checkpoint counter.

Unit

ELogging 103

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure ELogResetCheckPoint;
```

Remarks

Resets checkpoint counter, so the following calls to ELogAddCheckPoint 364 start over.

Examples

Code (Delphi)

```
ELogResetCheckPoint;

// do something dangerous
ELogAddCheckPoint;

// do something dangerous
ELogAddCheckPoint;

// do something dangerous
ELogAddCheckPoint;
```

See also

- ELogAddCheckPoint Function 364

10.65 ELogWarning

Adds a warning to log file.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogWarning(
   const AMsg: String
);
```

Parameters

AMsg [in]

A warning text.

Remarks

Same as ELog 360, but adds a warning instead of a normal text message.

Examples

Code (Delphi)

```
ELogWarning('This could be wrong');
```

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
ELog.Category('My Category').LogWarning('This could be wrong');
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogWarning('This could be
```

See also

- ELog Function 360
- ELogNote Function 377
- ELogError Function 371
- ELogReminder Function 380

10.66 ELogWindowHandle

ELog for window handle.

Unit

ELogging 103

Syntax

Code (Delphi)

```
procedure ELogWindowHandle(
   const AMsg: String;
   const AValue: HWND
);
```

Parameters

AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks

Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)

```
ELogWindowHandle('Window', Wnd);
```

Code (Delphi)

```
ELog.Category('My Category').LogWindowHandle('Window', Wnd);
```

Code (Delphi)

```
ELog.Category('My Category').FontColor(clRed).LogWindowHandle('Window',
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also
- ELog Function 360

10.67 ELogWinError

ELog for system error codes.

Unit
ELogging 103

Syntax

Code (Delphi)
```
procedure ELogWinError(
   const AMsg: String;
   const ALastError: TSysErrorCode 737 = TSysErrorCode(0
));
```

Parameters
AMsg [in]
A message to log.

AValue [in]
A value to log.

Remarks
Same as ELog 360. The separate function is required to avoid issues with overloads resolving.

Examples

Code (Delphi)
```
ELogWinError('GetLastError', GetLastError);
```

Code (Delphi)
```
ELog.Category('My Category').LogWinError('GetLastError', GetLastError);
```

Code (Delphi)
```
ELog.Category('My Category').FontColor(clRed).LogWinError('GetLastError'
```

See also
- ELog Function 360

10.68 EnableMemLeaksRegistering

Enables memory leaks checks.

Unit
EMemLeaks 109

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
procedure EnableMemLeaksRegistering;
```

Remarks

Enables back registering of memory leaks which was previously disabled with DisableMemLeaksRegistering 354.

Caution!

`DisableMemLeaksRegistering`/`EnableMemLeaksRegistering` can not be called recursively. If you make two calls to `DisableMemLeaksRegistering` followed to one call to `EnableMemLeaksRegistering` - then registering of memory leaks will be enabled.

See also

- DisableMemLeaksRegistering Function 354
- IsMemLeaksEnabled Function 461

10.69 EncodeConfigValue

Low-level routine to encode data before saving to configuration storage.

Unit

EConfig 63

Syntax

```
Code (Delphi)
function EncodeConfigValue(
   const AValue: String
): String;
```

Parameters

AValue

A data to encode.

Return value

Encoded data. Exact encoded format is unspecified.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

Encodes data by escaping special characters (like #0, line breaks, trailing spaces, etc).

Usually you don't need to use this function, because you're encouraged to use ConfigWriteString 342 or it's

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

wrapper. These ConfigWriteXYZ functions perform such encoding automatically.

See also

- DecodeConfigValue 352
- ConfigWriteString 342

10.70 EncryptBufferFree

Releases the encryption buffer.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure EncryptBufferFree(
   var AData: TEncryptBuffer 671
);
```

Parameters

AData [in]

Encryption buffer to release.

Remarks

This function will erase content of AData buffer (zero memory), then release the buffer, and set AData.pbData to nil, AData.cbData to 0.

Warning

AData buffer **must** be allocated via AllocMem function or be returned by any function from EEncrypt 80 unit.

Examples

Code (Delphi)

```
var
   Buffer: TEncryptBuffer;
begin
   // Allocate new buffer (600 bytes)
   Buffer.cbData := 600;
   Buffer.pbData := AllocMem(EncryptBuffer.cbData);
   try
      // Work with Buffer
   finally
      // Release buffer when done
      EncryptBufferFree(Buffer); // - same as SecureFree(Buffer);
   end;
end;
```

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
var
   Buffer: TEncryptBuffer;
   S: RawByteString;
begin
   S := 'something';

   // Point buffer to S
   Buffer.cbData := Length(S);
   Buffer.pbData := Pointer(S);

   // Work with Buffer

   // No need to release Buffer
   // (we did not allocate memory for buffer)
end;
```

See also

- EncryptBufferFree Function 385

10.71 EndThreadEx

Terminates the execution of a thread.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure EndThreadEx(
   const AExitCode: Integer = 0
);
```

Parameters

AExitCode [in, opt]
An optional thread exit code (thread termination status).

Return value

The function does **NOT** return.

Remarks

Warning

We do not recommend to use any of the following functions: `EndThreadEx`, `EndThread`, `ExitThread`. Just exit from thread's proc normally to ensure proper cleanup of allocated resources. Using any of the mentioned functions can cause memory leaks if you don't know what you are doing.

You can use this function instead of `EndThread` function inside threads created with `BeginThreadEx`.

`EndThreadEx` notifies EurekaLog about thread's termination and terminates the execution of a thread with the specified `ExitCode`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note
On Windows, `EndThreadEx` calls the `ExitThread` WinAPI with the specified `ExitCode`.

See also
- BeginThreadEx Function 325

10.72 EnsureThreadStack

Sets the minimum size of the stack associated with the calling thread or fiber that will be available during any stack overflow exceptions. This is useful for handling stack overflow exceptions; the application can safely use the specified number of bytes during exception handling.

Unit
ELowLevel 105

Syntax

Code (Delphi)
`function EnsureThreadStack: Boolean;`

Return value
If the function succeeds, the return value is `True`. If the function fails, the return value is `False`. To get extended error information, call `GetLastError`.

Remarks
You can specify the reserved stack space by setting the `_ThreadStackGuarantee` global variable. The function will do nothing if `_ThreadStackGuarantee` is zero or less than the current size.

If the function is successful, the application can handle possible `EStackOverflow` exceptions using structured exception handling. To resume execution after handling a stack overflow, you must perform certain recovery steps. Call the _resetstkoflwtstkoflw function.

To set the stack guarantee for a fiber, you must first call the `SwitchToFiber` function to execute the fiber. After you set the guarantee for this fiber, it is used by the fiber no matter which thread executes the fiber.

See also
- _ThreadStackGuarantee
- _resetstkoflw

10.73 EnumSenders

Returns list of registered send method class names.

Unit
ESend 120

Syntax

Code (Delphi)
`procedure EnumSenders(`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const ASenders: TStrings
);
```

Parameters

ASenders [out]
List of class names.

Remarks

Functions returns list of registered send method class names. Send method class registration is performed by RegisterSender 520 and RegisterSenderFirst 520 functions.

See also

- RegisterSender 520
- RegisterSenderFirst 520
- FindSender 401

10.74 EurekaAllocMem

AllocMem allocates a memory block and initializes each byte to zero.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaAllocMem(
   Size: TIntForRealloc
): Pointer;
```

Parameters

Size [in]
A size of memory block to allocate (in bytes).

Return value

A pointer to allocated memory block of `Size` bytes. Memory block is zero-initialized.

Remarks

`EurekaAllocMem` allocates a block of the given `Size` on the heap, and returns the address of this memory. Each byte in the allocated buffer is set to zero. To dispose of the buffer, use EurekaFreeMem 389. If there is not enough memory available to allocate the block, an leOutOfMemory error is raised.

Note

If the memory does not need to be zero-initialized, you can use EurekaGetMem 390.

Note

Do not call `EurekaAllocMem` manually. This function will be called automatically when you call AllocMem function when EurekaLog's memory filter is installed.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Important!

`EurekaAllocMem` usually never returns `nil` nor raise exceptions. Memory allocation errors are considered to be fatal. EurekaLog will call MemLeaksError event handler to hanle memory allocation error and terminate the application. See also: MemLeaksErrorsToIgnore global variable. You can use an explicit call to EurekaTryGetMem 394 function to allocate memory with ignoring errors.

See also

- EurekaGetMem Function 390
- EurekaTryGetMem Function 394
- EurekaFreeMem Function 389
- EurekaReallocMem Function 391
- IsMemLeaksEnabled Function 461
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

10.75 EurekaFreeMem

Frees a memory block previously allocated with EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaFreeMem(
    P: Pointer
): Integer;
```

Parameters

P [in]

A pointer to memory block returned by EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.

Return value

Return value is always 0.

Remarks

Use this procedure to dispose of a memory block obtained with EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.

Note

Do not call `EurekaFreeMem` manually. This function will be called automatically when you call FreeMem function when EurekaLog's memory filter is installed.

See also

- EurekaGetMem Function 390
- EurekaTryGetMem Function 394
- EurekaAllocMem Function 388

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EurekaReallocMem Function 391

10.76 EurekaGetMem

Allocates a memory block via EurekaLog memory filter.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaGetMem(
   Size: PtrInt
): Pointer;
```

Parameters

Size [in]

A size of memory block to allocate (in bytes).

Return value

A pointer to allocated memory block of `Size` bytes. Memory block is not zero-initialized.

Remarks

`EurekaGetMem` allocates a block of the given `Size` on the heap, and returns the address of this memory. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use EurekaFreeMem 389. If there is not enough memory available to allocate the block, an leOutOfMemory error is raised.

Note

If the memory block must be initialized with zero, you can use EurekaAllocMem 388.

Note

Do not call `EurekaGetMem` manually. This function will be called automatically when you call GetMem function when EurekaLog's memory filter is installed.

Important!

`EurekaGetMem` usually never returns `nil` nor raise exceptions. Memory allocation errors are considered to be fatal. EurekaLog will call MemLeaksError event handler to hanle memory allocation error and terminate the application. See also: MemLeaksErrorsToIgnore global variable. You can use an explicit call to EurekaTryGetMem 394 function to allocate memory with ignoring errors.

See also

- EurekaTryGetMem Function 394
- EurekaAllocMem Function 388
- EurekaFreeMem Function 389
- EurekaReallocMem Function 391
- IsMemLeaksEnabled Function 461
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.77 EurekaReallocMem

Resizes a memory block.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaReallocMem(
   P: Pointer;
   Size: PtrInt
): Pointer;
```

Parameters

P [in]

A pointer to memory block returned by EurekaGetMem 390, EurekaTryGetMem 394 or EurekaAllocMem 388.

Size [in]

New size of P in bytes. `Size` can be larger or smaller than original size of P. If `Size` is larger than old size, then the memory block will be expanded (with non initialized memory). If `Size` is smaller than old size, then the memory block will be truncated.

Return value

A pointer to allocated memory block of `Size` bytes. This value may or may not be equal to P. Old content of P is preserved. If `Size` is larger than old size of P, then newly added memory is not zero-initialized.

Remarks

`EurekaReallocMem` reallocates a memory block. `EurekaGetMem` allocates a block of the given `Size` on the heap. If you have to change the size of this memory block, call `EurekaReallocMem`, passing the existing memory block pointer in P and the revised block size in `Size`. If `EurekaReallocMem` can't expand the memory block pointed to by P, it frees the referenced memory and copies the values to the newly allocated memory that is returned. The content of the newly allocated memory is not set to zero.

Note

Do not call `EurekaReallocMem` manually. This function will be called automatically when you call ReallocMem function when EurekaLog's memory filter is installed.

Important!

`EurekaReallocMem` usually never returns `nil` nor raise exceptions. Memory allocation errors are considered to be fatal. EurekaLog will call MemLeaksError event handler to hanle memory allocation error and terminate the application. See also: MemLeaksErrorsToIgnore global variable. You can use an explicit call to EurekaTryGetMem 394 function to allocate memory with ignoring errors.

See also

- EurekaGetMem Function 390
- EurekaTryGetMem Function 394
- EurekaAllocMem Function 388

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EurekaFreeMem Function 389
- IsMemLeaksEnabled Function 461
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

10.78 EurekaRegisterExpectedMemoryLeak

Register an expected memory leak.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaRegisterExpectedMemoryLeak(
   P: Pointer
): Boolean;
```

Parameters

P [in]

A pointer to memory block of expected leak.

Return value

Indicate if leak was registered successfully. A `False` value usually indicate that this block was already registered.

Remarks

This function registers a particular memory block to be excluded from memory leaks report. This feature can be used to skip reporting for "leaks by design" or for bugs in 3rd party code over which you have no control.

This function is used to exclude already allocated blocks. Thus, this function is usually used to register leaks from 3rd party code. To register expected leaks in your own code - you can use either this function or DisableMemLeaksRegistering function 354.

Tip

While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeak 395 function to unregister memory block from excluded leaks.

Important!

You must pass a pointer to the beginning of the memory block. Please note that dynamic arrays and strings do not point to the beginning of allocation. Use EurekaRegisterExpectedMemoryLeakDynArray 393 and EurekaRegisterExpectedMemoryLeakStr 393 for dynamic arrays and strings instead of this function.

See also

- EurekaUnregisterExpectedMemoryLeak Function 395
- EurekaRegisterExpectedMemoryLeakDynArray Function 393
- EurekaRegisterExpectedMemoryLeakStr Function 393
- DisableMemLeaksRegistering Function 354

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.79 EurekaRegisterExpectedMemoryLeakDynArray

Register an expected memory leak (variant for dynamic arrays).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaRegisterExpectedMemoryLeakDynArray(
    P: Pointer
): Boolean;
```

Parameters

P [in]

A dynamic array casted to pointer.

Return value

Indicate if leak was registered successfully. A `False` value usually indicate that this block was already registered.

Remarks

This function is the same as EurekaRegisterExpectedMemoryLeak function 392, but it accepts only pointers to dynamic arrays.

Tip

While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeakDynArray 395 function to unregister memory block from excluded leaks.

See also

- EurekaUnregisterExpectedMemoryLeakDynArray Function 395
- EurekaRegisterExpectedMemoryLeak Function 392

10.80 EurekaRegisterExpectedMemoryLeakStr

Register an expected memory leak (variant for strings).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaRegisterExpectedMemoryLeakStr(
    P: Pointer
): Boolean;
```

Parameters

P [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

A dynamic array casted to pointer.

Return value

Indicate if leak was registered successfully. A `False` value usually indicate that this block was already registered.

Remarks

This function is the same as EurekaRegisterExpectedMemoryLeak function 392, but it accepts only pointers to strings (ANSI or Unicode, but not WideStrings).

Tip
While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeakStr 396 function to unregister memory block from excluded leaks.

See also

- EurekaUnregisterExpectedMemoryLeakStr Function 396
- EurekaRegisterExpectedMemoryLeak Function 392

10.81 EurekaTryGetMem

Allocates a memory block via EurekaLog memory filter (w/o error checking).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaTryGetMem(
   Size: PtrUInt 609
): Pointer;
```

Parameters

`Size` [in]

A size of memory block to allocate (in bytes).

Return value

A pointer to allocated memory block of `Size` bytes (not initialized to zero) or `nil` if memory error occurs.

Remarks

`EurekaTryGetMem` allocates a block of the given `Size` on the heap, and returns the address of this memory. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use EurekaFreeMem 389. If there is not enough memory available to allocate the block, `nil` is returned.

Note
Unlike EurekaGetMem 390, `EurekaTryGetMem` function can be called manually.

See also

- EurekaGetMem Function 390
- EurekaAllocMem Function 388

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- EurekaFreeMem Function 389
- EurekaReallocMem Function 391
- IsMemLeaksEnabled Function 461
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

10.82 EurekaUnregisterExpectedMemoryLeak

Removes memory block from excluded leaks.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeak(
    P: Pointer
): Boolean;
```

Parameters

P [in]

A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeak function 392.

Return value

`True` - expected leak was unregistered successfully. `False` - leak was not unregistered (e.g. leak was already unregistered earlier, no such leak was registered, or EurekaRegisterExpectedMemoryLeak function 392 returned `False`).

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeak Function 392

10.83 EurekaUnregisterExpectedMemoryLeakDynArray

Removes memory block from excluded leaks (variant for dynamic arrays).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeakDynArray(
    P: Pointer
): Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

P [in]
A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeakDynArray function 393.

Return value

`True` - expected leak was unregistered successfully. `False` - leak was not unregistered (e.g. leak was already unregistered earlier, no such leak was registered, or EurekaRegisterExpectedMemoryLeakDynArray function 393 returned `False`).

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeakDynArray Function 393

10.84 EurekaUnregisterExpectedMemoryLeakStr

Removes memory block from excluded leaks (variant for strings).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeakStr(
    P: Pointer
): Boolean;
```

Parameters

P [in]
A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeakStr function 393.

Return value

`True` - expected leak was unregistered successfully. `False` - leak was not unregistered (e.g. leak was already unregistered earlier, no such leak was registered, or EurekaRegisterExpectedMemoryLeakStr function 393 returned `False`).

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeakStr Function 393

10.85 ExceptionManager

Exception manager.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EExceptionManager 93

Syntax

Code (Delphi)
```
function ExceptionManager: TExceptionManager 232;
```

Return value
Instance of exception manager.

Remarks
This is global access method for exception manager. Use it to invoke routines of exception manager. Do not create instances of `TExceptionManager` class.

Exception manager instance is created and deleted automatically. You don't need to manually control life-time of this object.

See TExceptionManager 232 for more information.

Note

This function is the same as TExceptionManager 232.Instance 233.

See also
- TExceptionManager class 232
- TExceptionManager.Instance class method 233

10.86 ExceptionMessage

Retrieves error message for an exception.

Unit
EBase 57

Syntax

Code (Delphi)
```
function ExceptionMessage(
   const E: TObject
): String;
```

Parameters
E [in]
Exception object to get its `.Message` property.

Return value
Error message for the specified exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Use this function to get error message for any given exception, even if it is not `Exception` class.

See also

- SetExceptionMessage Function 551

10.87 ExpandEnvVars

Expands environment-variable strings.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function ExpandEnvVars(
   const ASource: String;
   AGetEnvVariable: TGetEnvVariableFunc = nil
): String; overload;

function ExpandEnvVars(
   const ASource: String;
   AGetEnvVariable: TGetEnvVariableMethod
): String; overload;
```

Parameters

`ASource` [in]

A buffer that contains one or more environment-variable strings in the form: %*variableName*%. For each such reference, the %*variableName*% portion is replaced with the current value of that environment variable.

Case is ignored when looking up the environment-variable name. If the name is not found, the %*variableName*% portion will be unchanged.

`AGetEnvVariable` [in, optional]

An optional environment variable converter.

There are overloads for procedural and method converter.

This function is used to get value of variables. You can supply your own converter to support more variables or change value of existsing ones.

If you don't specify this argument - the default converter will be used (GetEnvVariable 413).

Return value

`ASource` buffer with expanded environment variables.

Remarks

Expands environment-variable strings and replaces them with the values defined for the current user. You can override variables by supplying the second argument (`AGetEnvVariable`)

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This function is similar to ExpandEnvironmentStrings.

Note

This routine is often replaced in EurekaLog classes to supply extra variables. For example, exception informational class adds support for %_ExceptMsg% variable.

Examples

Code (Delphi)

```
ExpandEnvVars('Windows dir: %WinDir%; non-existent var: %something%') =
'Windows dir: C:\Windows\; non-existent var: %something%'
```

See also

- Environment variables
- GetEnvVariable 413
- GetEnvVariable 413

10.88 FindDialogClass

Searches dialog class by name.

Unit

EDialog 72

Syntax

Code (Delphi)

```
function FindDialogClass(
   const AClassName: String
): TBaseDialogClass 611;
```

Parameters

AClassName [in]

Name of dialog class. For example: 'TMessageBoxDialog'.

Return value

Registered dialog class or `nil` if no class was found.

Remarks

Functions looks for registered dialog class with the specified name and return first found class. Dialog class registration is performed by RegisterDialogClass 502 and RegisterDialogClassFirst 503 functions.

See also

- RegisterDialogClass 502
- RegisterDialogClassFirst 503

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.89 FindLowAddress

Tool function to obtain starting address to disassemble code block around known address.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
function FindLowAddress(
   const AExceptionAddress: Pointer;
   const ALines: Integer
): Pointer;
```

Parameters

AExceptionAddress [in]
A known code pointer.

ALines [in]
How many lines (instructions) should be before AExceptionAddress.

Return value

A pointer to beginning of some CPU instruction. The returned instruction will be exactly ALines instructions before AExceptionAddress.

Remarks

This is auxilary function for DisasmFunction 355. It's used to find some starting address to begin disassebling code block. Usually you know exception address (i.e. some pointer to code), but don't know exact starting address of the function.

Note

When this function is used with DisasmFunction 355, it's recommended that FindLowAddress's ALines argument equals to DisasmFunction's ALines argument.

10.90 FindScreenshotRect

Determinates screen area occupied by the specified object.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
function FindScreenshotRect(
   const AScreenshotMode: TSendScreenshot 727
): TRect;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AScreenshotMode [in]
Specifies a screen area to locate.

Return value

Absolute screen (desktop) coordinates of the `AScreenshotMode`. Or empty rectangle on failure.

Remarks

This is a auxiliary routine which helps determinate what part of the screen (desktop) should be included in the screenshot.

The results of this function can be used to call Screenshot 541 or ScreenshotInit 543 routines.

See also

- Screenshot Routine 541
- ScreenshotInit Routine 543

10.91 FindSender

Searches send method class by name.

Unit

ESend 120

Syntax

Code (Delphi)

```
function FindSender(
   const ASenderClassName: String
): TELUniversalSenderClass 669;
```

Parameters

ASenderClassName [in]
Name of send method class. For example: 'TELTrackerMantisSender'.

Return value

Registered send method class or `nil` if no class was found.

Remarks

Functions looks for registered send method class with the specified name and return first found class. Send method class registration is performed by RegisterSender 520 and RegisterSenderFirst 520 functions.

See also

- RegisterSender 520
- RegisterSenderFirst 520
- FindSender 401

10.92 FreezeNotifyStillAlive

Lets EurekaLog know that your main thread is still working.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
procedure FreezeNotifyStillAlive;
```

Remarks

This function will temporarily stop freeze detection without shutdowning background freeze detection thread.

EurekaLog tries to send messages to main thread from time to time. If your main thread is busy or hang - it will not process the messages, therefore EurekaLog will trigger "The thread appears to be frozen" error. Calling this function will skip a single check for main thread's responsivness, so EurekaLog will know that your thread is OK, even though it is not processing messages.

Warning

`FreezeNotifyStillAlive` should be called frequently. Imagine you have notified our freeze checks that your thread is alive. OK, sure: our checks will receive the message and skip a single check. Next, your thread will do something long. Now you are not sending "I am still alive" messages. Our checks will trigger immediately on the next closest check, because your thread is not responding for messages nor we did not receive "I'm still alive" notification.

Tip

For the above reason we recommend to use PauseFreezeCheck 497 instead.

It is usually better to just let main thread process messages instead of calling this function. If it is not possible - consider offloading work into a background thread. You can even do this:

Code (Delphi)

```
Thread := TThread.Create(True);
Thread.Start;
if WaitForSingleObject(Thread.Handle, YourMaxExecutionTimeInMS) = WAIT_T
   RaiseFreezeException(Thread.ThreadID, Thread.Handle);
```

Important!

`FreezeNotifyStillAlive` can be called only by main thread.

See also

- RaiseFreezeException Function 499
- PauseFreezeCheck Function 497
- IsFreezeCheckInstalled Function 458
- DoneCheckFreeze Function 357
- TFreezeThread.ChecksPaused Property 247

10.93 GetActiveFormClass

Gets window class of active window.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ESysInfo 134

Syntax

Code (Delphi)
```
function GetActiveFormClass(
   const AWnd: HWND = 0
): String;
```

Return value
If the function succeeds and there is active window, the return value is class of active window.

Sample output: 'TAppBuilder'.

If the function succeeds but there is no active window, the return value is empty string.

If the function fails, the return value is error message.

Tip

Return value never contains line breaks.

Remarks
This function is combination of GetClassName (GetForegroundWindow)

See also
- GetActiveFormText 403
- GetControlClass 409
- GetControlText 410

10.94 GetActiveFormText

Gets caption of active window.

Unit
ESysInfo 134

Syntax

Code (Delphi)
```
function GetActiveFormText(
   const AWnd: HWND = 0
): String;
```

Return value
If the function succeeds and there is active window, the return value is caption of active window.

Sample output: 'XMLDocToXML - Embarcadero RAD Studio 2010 - ESysInfo'

If the function succeeds but there is no active window, the return value is empty string.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

If the function fails, the return value is error message.

Remarks

This function is combination of WM_GETTEXT message from GetForegroundWindow

Note
This function uses a timeout of 100 ms. If active window is hung and not responding - function will fail after timeout. Therefore it's safe to call this function, no deadlock is possible.

Tip
Return value nevers contain line breaks.

See also

- GetActiveFormClass 402
- GetControlClass 409
- GetControlText 410

10.95 GetApplicationDescription

Gets application description.

Unit

ESysInfo 134

Syntax

```
Code (Delphi)
function GetApplicationDescription: String;
```

Return value

Description of .exe file or empty string if it's not available.

Sample output: 'bds.exe - (Embarcadero RAD Studio for Windows)'

Remarks

This function extracts description of .exe file from version information. If such information is present - function returns "File description" field with .exe file name. Otherwise empty string is returned.

This function is simple combination GetModuleDescription(PrimaryInstance 498)

See also

- GetModuleDescription
- GetApplicationName 404
- GetExceptionModuleName 419
- PrimaryInstance 498
- CurrentInstance 348

10.96 GetApplicationName

Gets application name.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ESysInfo 134

Syntax

Code (Delphi)
```
function GetApplicationName: String;
```

Return value
Name of .exe file or empty string if it's not available.

Sample output: 'Embarcadero RAD Studio for Windows'

Remarks
This function extracts description of .exe file from version information. If such information is present - function returns "File description" field. Otherwise empty string is returned.

This function is simple combination GetModuleDescriptionAndVersion(PrimaryInstance 498)

See also
- GetModuleDescriptionAndVersion
- GetApplicationDescription 404
- GetExceptionModuleName 419
- PrimaryInstance 498
- CurrentInstance 348

10.97 GetApplicationRect

Gets bounding rectangle for GUI application.

Unit
ESysInfo 134

Syntax

Code (Delphi)
```
function GetApplicationRect(
   const APID: DWord
): TRect;
```

Parameters
APID [in]
Process ID of running application.

Return value
Bounding rectangle which includes all visible windows of the specified application.

If there is no visible windows belonging to the application - an empty region will be returned (0, 0, 0, 0).

Remarks
Function enumerates all windows of the specified application and find the smallest possible rectangle area

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

which will include all such windows. The resulting area is returned.

This function is used when taking screenshot of desktop with "Capture my application's windows only" option enabled - to determinate region of desktop to take screenshot from.

See also
- Screenshot routines 119

10.98 GetApplicationWnd

Returns application window (if applicable).

Unit
EAppType 54

Syntax

Code (Delphi)
```
function GetApplicationWnd: HWND;
```

Return value
Handle of application's window.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

By default it's 0.

Plugin can specify a way to get handle by installing a _GetAppWnd handler.

Note

Typically this is Application.Handle.

See also
- GetMainWnd 424
- GetTaskBarWnd 429
- GetParentWnd 426
- _GetAppWnd
- IsAppType 442

10.99 GetAsmSize

Returns size of a single machine opcode.

Unit
EDisAsm 78

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
function GetAsmSize(
   const AStart: Pointer;
   var ASize: Byte
): Boolean;
```

Parameters

AStart
A pointer to code.

ASize
(out) A size of instruction at AStart (in bytes).

Return value

True if operation was successful. False - otherwise.

Exceptions

System.SysUtils.EAccessViolation	Invalid pointer or invalid code.

Remarks

GetAsmSize tries to determinate size of one machine instruction in bytes.

This function is used as base of GetFunctionSize. It's also used in hooks and disassembler-related code.

Note

If AStart points to JMP-stub, GetAsmSize will return size of the stub, not the size of first function instruction.

See also

- GetFunctionSize
- ConvertAddress 343

10.100 GetCompanyName

Gets company name.

Unit

ESysInfo 134

Syntax

```
function GetCompanyName: String;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Company name or empty string.

Example value: 'Litware Ltd.'

Remarks

This function checks the following registry keys for company information and returns the first suitable value:

- '\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION'
- '\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION'
- '\SOFTWARE\MICROSOFT\MS SETUP (ACME)\USER INFO'

See also

- GetComputerName 409
- GetUserEMail 432
- GetUserFullName
- GetUserName

10.101 GetCompilationDate

Returns the compilation date of the specified executable module.

Unit

EModules 114

Syntax

Code (Delphi)

```
function GetCompilationDate(
   const AModule: HINST;
   const ALocalTime: Boolean;
   out ADate: TDateTime
): Boolean;
```

Parameters

AModule [in]

A handle of any loaded module in the current process. Typical value is HInstance.

ALocalTime [in]

Pass `True` if you want to get compilation date in your current time zone; pass `False` if you want to get compilation date in GMT/UTC.

ADate [out]

The compilation date of the specified module.

Return value

`True` if function was able to obtain the compilation date. `False` otherwise. Function should always succeed if called for EurekaLog-enabled module. Function could fail if invalid handle is passed, if module is not EurekaLog-enabled module, and module does not have valid date in its headers.

Remarks

This function reads post-processing date (almost the same as compilation date) from EurekaLog's embedded data for the module. If module does not have EurekaLog embedded - then the function tries to read date from module's PE headers.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note
If module does not have EurekaLog injected - then it may not always be possible to retrieve date in the correct time zone.

See also
- __DATE__ function
- __TIME__ function

10.102 GetComputerName

Gets computer name user name.

Unit
ESysInfo 134

Syntax

Code (Delphi)
`function GetComputerName: String;`

Return value
Current computer name or empty string.

Example value: 'WORKSTATION1'

Remarks
Retrieves the NetBIOS name of the local computer. This name is established at system startup, when the system reads it from the registry. The behavior of this function can be affected if the local computer is a node in a cluster. If the caller is running under a client session, this function returns the server name.

Note
This is a wrapper for GetComputerName function.

See also
- GetCompanyName 407
- GetUserEMail 432
- GetUserFullName
- GetUserName

10.103 GetControlClass

Gets class of active control.

Unit
ESysInfo 134

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function GetControlClass(
   const AWnd: HWND = 0
): String;
```

Return value

Class of active control.

Sample output: 'TEdit'

Remarks

This function is combination of GetClassName (GetFocus)

See also

- GetActiveFormClass 402
- GetActiveFormText 403
- GetControlText 410

10.104 GetControlText

Gets text of active control.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetControlText(
   const AWnd: HWND = 0
): String;
```

Return value

Text of active (focused) single line text control. Empty string if there is no focused single line control.

Sample output: 'Some text'

Remarks

This function is combination of WM_GETTEXT message from GetFocus

Note

This function uses a timeout of 100 ms. If active window is hung and not responding - function will return empty string after timeout. Therefore it's safe to call this function, no deadlock is possible.

Note

This function will not return clear text from password controls. Passwords will be masked by password characters (typically - *).

See also

- GetActiveFormClass 402
- GetActiveFormText 403

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- GetControlClass 409

10.105 GetCurrentModuleCompilationDate

Gets compilation time of current module.

Unit
ESysInfo 134

Syntax

Code (Delphi)

```
function GetCurrentModuleCompilationDate: TDateTime;
```

Return value
Compilation date-time of current module in local time or 0 if it's not available.

Remarks
This function is simple combination of GetCompilationDate 408(PrimaryInstance 498).

Please see GetCompilationDate 408 for limitations of this function.

See also
- GetCompilationDate 408
- PrimaryInstance 498
- CurrentInstance 348

10.106 GetCurrentUserPrivileges

Gets description of user privileges.

Unit
ESysInfo 134

Syntax

Code (Delphi)

```
function GetCurrentUserPrivileges: String;
```

Return value
Status of enabled and disabled system privileges of the current security token. Result string will be multi-line values - one priviledge and its status per line.

Example return value (multi-line):

SeShutdownPrivilege - ON

SeChangeNotifyPrivilege - ON

SeUndockPrivilege - OFF

SeIncreaseWorkingSetPrivilege - OFF

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

SeTimeZonePrivilege - OFF

Remarks

Function opens current process and ask for security token, then enumerates privileges listed in this token and collect status of all included privileges.

10.107 GetDiskInfo

Gets total and free disk space size.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetDiskInfo(
   out AFreeDisk, ATotalDisk: UInt64
): Boolean;
```

Parameters

AFreeDisk [out]
Free disk space in bytes.

ATotalDisk [out]
Total disk space in bytes.

Return value

If the function succeeds, the return value is `True`.

If the function fails, the return value is `False`. To get extended error information, call GetLastError.

Remarks

Function returns disk space size information for disk, which contains current executable module (returned by CurrentInstance 348).

See also

- GetFreeDisk
- GetTotalDisk
- CurrentInstance 348
- PrimaryInstance 498

10.108 GetDisplayDPI

Gets screen DPI.

Unit

ESysInfo 134

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function GetDisplayDPI: Integer;
```

Return value
DPI of desktop.

Example values: 96, 120, etc.

Remarks
Basically, this is GetDeviceCaps(GetDC(0), LOGPIXELSY).

See also
- GetDisplayMode 413

10.109 GetDisplayMode

Gets screen mode.

Unit
ESysInfo 134

Syntax

Code (Delphi)

```
function GetDisplayMode: String;
```

Return value
Screen resolution and color depth.

Example value: '1680 x 1050, 32 bit'

Remarks
Function returns settings of primary monitor.

Note

Desktop pixel size may be larger in multi-monitor configurations. This function counts only primary monitor, all other monitors are ignored.

See also
- GetDisplayDPI 412

10.110 GetEnvVariable

Expands single environment variable.

Unit
ESysInfo 134

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function GetEnvVariable(
   const AName: String
): String;
```

Parameters

AName [in]
Name of variable without %.

Return value

Variable value or empty string if variable wasn't defined.

Remarks

This is default implementation for variable name convertor for ExpandEnvVars 398 function.

This default implementation supports all standard system and user environment variables and also adds the following additional EurekaLog pseudo-variables:

- CSIDL_*XYZ*
- FOLDERID_*XYZ*
- _ThisModulePath
- _ThisModuleName
- _MainModulePath
- _MainModuleName
- _LineBreak
- CD
- Date
- DateFmt
- Time
- TimeFmt
- Random

Note
This routine is often replaced in EurekaLog classes to supply extra variables. For example, exception informational class adds support for %_ExceptMsg% variable.

See also

- ExpandEnvVars 398

10.111 GetEurekaLogLabel

Gets full EurekaLog version.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetEurekaLogLabel: String;
```

Return value

Actual EurekaLog version information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Example values:

'EurekaLog 7.0.0.177 Trial'

'EurekaLog 7.0.0.177 Professional'

'EurekaLog 7.0.0.177 Enterprise'

'EurekaLog 7.0.0.177 Registered' (for not installed EurekaLog)

'EurekaLog 7.0.0.177 RC 2 Enterprise (BETA)'

Remarks

This function returns information about EurekaLog. Thus, its behaviour is different depending on EurekaLog status.

If EurekaLog is installed - function reads information about EurekaLog's version from the registry.

If EurekaLog is not installed - function returns build-in information.

Note
Result of this function equals GetEurekaLogVersion 418 + GetEurekaLogLicense 415 + GetEurekaLogStatus 417.

Note
If you want version information about EurekaLog in current module - there is no need to call any function. Such information is always available in EEurekaLogVersion 755 and EEurekaLogLicense 754 constants.

See also

- GetEurekaLogVersion 418
- GetEurekaLogLicense 415
- GetEurekaLogStatus 417
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 755
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

10.112 GetEurekaLogLicense

Gets EurekaLog license.

Unit

ESysInfo 134

Syntax

```
function GetEurekaLogLicense: String;
```

Return value

Actual EurekaLog license information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Example values:

'Trial'

'Professional'

'Enterprise'

'Registered' (for not installed EurekaLog)

Remarks

This function returns information about EurekaLog. Thus, its behaviour is different depending on installed EurekaLog.

If EurekaLog is installed - function reads information about the installed EurekaLog from the registry.

If EurekaLog is not installed - function returns build-in information.

Note
If you want license information about EurekaLog in current module - there is no need to call any function. Such information is always available in EEurekaLogLicense 754 constant.

See also

- GetEurekaLogLicenseType 416
- GetEurekaLogLabel 414
- GetEurekaLogVersion 418
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 755
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

10.113 GetEurekaLogLicenseType

Gets EurekaLog license type.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetEurekaLogLicenseType: String;
```

Return value

Actual EurekaLog license type information.

Example values:

'Single Developer'

'MVP'

'Company'

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

'Corporate'

Remarks

This function returns information about EurekaLog. Thus, its behaviour is different depending on installed EurekaLog.

If EurekaLog is installed - function reads information about the installed EurekaLog from the registry.

If EurekaLog is not installed - function returns build-in information.

See also

- GetEurekaLogLicense 415
- GetEurekaLogLabel 414
- GetEurekaLogVersion 418
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 755
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

10.114 GetEurekaLogPath

Obsolete function, do not use.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetEurekaLogPath: String;
deprecated'Use GetFolderEurekaLogAppData';
```

Remarks

Warning

This function is provided for backward compatibility only. It may be removed in the next version of EurekaLog.

Use GetFolderEurekaLog function instead.

See also

- GetFolderEurekaLog

10.115 GetEurekaLogStatus

Gets EurekaLog stable/beta information.

Unit

ESysInfo 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
function GetEurekaLogStatus: String;
```

Return value

Status of current EurekaLog sources.

Example values:

''

'(BETA)'

Remarks

Function returns build-in information. No registry is read.

See also

- GetEurekaLogLabel 414
- GetEurekaLogVersion 418
- GetEurekaLogLicense 415
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 755
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

10.116 GetEurekaLogVersion

Gets EurekaLog version.

Unit

ESysInfo 134

Syntax

```
Code (Delphi)
function GetEurekaLogVersion: String;
```

Return value

String representation of current EurekaLog's version.

Example value: '7.0.0.177'.

For beta versions, RC versions and other non-stable releases a suffix is added. Example: '7.0.0.177 RC 2'

Remarks

This function is the same as EEurekaLogVersion 755 constant.

represents EurekaLog's version number in the standard 4-number format 'A.B.C.D' plus optional suffix for non-stable releases.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- GetEurekaLogLabel 414
- GetEurekaLogLicense 415
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 755
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

10.117 GetExceptionModuleName

Gets description of exception's module.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetExceptionModuleName(
   const AAddr: Pointer
): String;
```

Parameters

AAddr [in]

Exception address. Usually it's ExceptAddr value or equivalent.

This address is used to determinate handle of module which contains code with the specified address.

Return value

Description of exception module file or empty string if it's not available.

Sample output: 'bds.exe - (Embarcadero RAD Studio for Windows)'

Remarks

This function extracts description of module file from version information. If such information is present - function returns "File description" field with module file name. Otherwise empty string is returned.

If exception was raised in .exe file - the return value will be equal to GetApplicationDescription 404.

This function is simple combination GetModuleDescription(FindHInstance(AAddr))

See also

- GetModuleDescription
- GetApplicationDescription 404
- GetExceptionModuleVersion 419

10.118 GetExceptionModuleVersion

Gets exception's module version.

Unit

ESysInfo 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function GetExceptionModuleVersion(
   const AAddr: Pointer
): String; overload;

function GetExceptionModuleVersion(
   const AModuleName: String
): String; overload;
```

Parameters

AAddr [in]

Exception address. Usually it's ExceptAddr value or equivalent.

This address is used to determinate handle of module which contains code with the specified address.

AModuleName [in]

Full file name to get version number.

Return value

File version of exception's module.

Example value: '1.0.0.0'.

Remarks

This function extracts version number of module file from version information. If such information is present - function returns "File version" field. Otherwise empty string is returned.

See also

- GetExceptionModuleName 419
- GetFileVersion

10.119 GetFileIcon

Gets executable file's icon.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetFileIcon(
   const ADPI: Integer;
   const AFileName: String;
   ALarge: Boolean;
   ASz: PSize;
   const AIconName: UnicodeString = '';
   const ALoadFromRCData: Boolean = False
): HICON;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

ADPI [in, optional]
DPI settings (96, 120, etc.). Pass 0 to use defaults.

AFileName [in]
Full executable file name (.exe, .dll, .bpl).

ALarge [in]
`True` - extract large icon, `False` - extract small icon.

Icon sizes are determinated from system settings and DPI. Usually it's values like 40x40 or 32x32 for large size and 20x20 or 16x16 for small size.

ASz [out, optional]
Actual icon size in pixels. Caller may use this value for rendering resulting icon.

This value is undefined if function fails.

Return value

If the function succeeds, the return value is default icon's handle of the specified file.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Creates handle to large or small icon extracted from the specified executable file, DLL, or package.

You must destroy all icons extracted by `GetFileIcon` by calling the DestroyIcon function.

10.120 GetFileSize

Gets file size.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetFileSize(
   const AFileName: String
): UInt64;
```

Parameters

AFileName [in]
Full file name.

Return value

If the function succeeds, the return value is size of the specified file in bytes.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Retrieves the size of the specified file, in bytes.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.121 GetFunctionSize

Returns size of the function in bytes.

Unit

EDisAsm 78

Syntax

Code (Delphi)

```
function GetFunctionSize(
   const AAddr: Pointer;
   const AMaxSize: Cardinal
): Cardinal;
```

Parameters

AAddr

A pointer to function.

AMaxSize

Maximum possible size to scan. Use this to limit scan area and prevent unlimited reading.

Return value

Size of function at `AAddr` in bytes. This value is always in range 0..`AMaxSize`.

Exceptions

System.SysUtils.EAccessViolation	Invalid address or invalid code.

Remarks

`GetFunctionSize` scans machine code at `AAddr` address up to `AAddr + AMaxSize`. It stops at first `ret` instrution and returns distance between `AAddr` and end of `ret`.

This function is used in hooks and disassembler-related code.

Note

Make sure that you pass a pointer to a real function. Not to some arbitrary code or function's stub.

Use to get pointer to real code from stub.

Warning

This is heuristic function. Results are not 100% reliable.

See also

- GetAsmSize
- ConvertAddress 343

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.122 GetLocationInfo

Returns human-readable information about code location.

Unit

EDebugInfo 68

Syntax

Code (Delphi)

```
function GetLocationInfo(
   const Addr: Pointer
): TELLocationInfo 662; overload;

function GetLocationInfo(
   const Addr: Pointer;
   var Info: TELLocationInfo 662
): Boolean; overload;
```

Parameters

Addr [in]
A pointer to a code (method, function, line, etc.)

Info [out]
A human-readable description of the code (unit, class, rouitne, line, offsets).

Return value

A human-readable description of the code (unit, class, rouitne, line, offsets), or success / failure state.

Remarks

Use this function to retrieve location information about code instruction. EurekaLog will load debug information (if any available) and look it up for the specified address.

See also

- GetLocationInfoStr function 423
- LocationToStr Function 478
- __MODULE__ function
- __UNIT__ function
- __FILE__ function
- __FUNCTION__ function
- __LINE__ function

10.123 GetLocationInfoStr

Returns human-readable information about code location.

Unit

EDebugInfo 68

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function GetLocationInfoStr(
   const Addr: Pointer
): String; overload;

function GetLocationInfoStr(
   const Addr: Pointer;
   out Info: String
): Boolean; overload;
```

Parameters

Addr [in]
A pointer to a code (method, function, line, etc.)

Info [out]
A human-readable description of the code (unit, class, routitne, line, offsets). For example: '(003BDD73) {Project1.exe} [007BDD73] Unit1.TForm1.Button1Click (Line 32, "Unit1.pas") + $5'

Return value

A human-readable description of the code (unit, class, routitne, line, offsets), or success / failure state. For example: '(003BDD73){Project1.exe} [007BDD73] Unit1.TForm1.Button1Click (Line 32, "Unit1.pas") + $5'

Remarks

This function is a simple combination of GetLocationInfo 423 and LocationToStr 478 functions.

See also

- GetLocationInfo function 423
- LocationToStr Function 478
- __MODULE__ function
- __UNIT__ function
- __FILE__ function
- __FUNCTION__ function
- __LINE__ function

10.124 GetMainWnd

Returns main window of application (if applicable).

Unit

EAppType 54

Syntax

Code (Delphi)

```
function GetMainWnd: HWND;
```

Return value

Handle of main application's window.

Remarks

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

By default it's 0.

Plugin can specify a way to get handle by installing a _GetMainWnd handler.

Note

Typically this is Application.MainForm.Handle.

See also

- GetTaskBarWnd 429
- GetApplicationWnd 406
- GetParentWnd 426
- _GetMainWnd
- IsAppType 442

10.125 GetModuleType

Returns type of loaded executable module, based on analyzing its file header.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function GetModuleType(
   const AInstance: HMODULE = 0
): TModuleType 709;
```

Parameters

AInstance [in, optional]

Module handle to analyze. Default (0) means current module.

Be careful with using 0 in packaged application. It's always better to explicitly specify required module.

Return value

Type of the AInstance module. It is TModuleType.mtProgram 710, TModuleType.mtLibrary 709 or TModuleType.mtPackage 710 on success and TModuleType.mtUnknown 710 on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This function analyzes PE header of loaded executable and return its type: EXE, DLL or BPL.

Function doesn't cache results and perform checking on each call.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

There are also simple wrappers for more easy access. For example, IsEXE 455, IsDLL 449 and so on. See EAppType 54 unit for more info.

See also

- TModuleType 709
- CurrentInstance 348

10.126 GetMXServerFromEmail

Retrieves mail server for e-mail address.

Unit

EDNS 79

Syntax

Code (Delphi)

```
function GetMXServerFromEmail(
   const EMail: String
): String;
```

Parameters

EMail [in]

A valid e-mail address, such as 'reports@example.com'.

Return value

Host or IP address of SMTP server (such as 'mail.example.com' or 'smtp.example.com'). Use this SMTP address to send e-mails to EMail e-mail address.

Exceptions

EDNS.EDNSError 151	Raised when DnsQuery function fails.

Remarks

This function sends a DNS query about host from e-mail address (e.g. "example.com"). DNS query asks about MX-entry.

10.127 GetParentWnd

Returns active or top-most window, which can be used as a parent window for modal dialogs.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function GetParentWnd: HWND;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value
Handle of an active/top-most window.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

Use this function to get window of application for functions like `Windows.MessageBox` or `ShellAPI.ShellExecute`, which require a window handle.

By default it's 0.

Plugin can specify a way to get handle by installing a _GetParentWnd handler.

Note
Typically this is Application.ActiveFormHandle.

See also
- GetMainWnd 424
- GetTaskBarWnd 429
- GetApplicationWnd 406
- _GetParentWnd
- IsAppType 442

10.128 GetPerformanceCount

Returns timestamp.

Unit
ELogging 103

Syntax

Code (Delphi)
```
function GetPerformanceCount: Int64;
```

Return value
A timestamp to be passed to ELogInterval 375 function.

Remarks
This function returns current date-time as timestamp. The return value of this function could be passed to ELogInterval 375 function.

Examples

Code (Delphi)
```
Start := GetPerformanceCount;
// your code
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
ELogInterval('Code took the following time', Start);
```

See also

- ELogInterval Function 375
- ELog Function 360

10.129 GetReadableSize

Verifies that the caller has specified access to the specified range of memory.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function GetReadableSize(
   const AAddress: Pointer;
   const ASize: Cardinal;
   const AAttributes: Cardinal = 0
): Cardinal;
```

Parameters

AAddress [in]
A pointer to the first byte of the memory block.

ASize [in]
The size of the memory block, in bytes. If this parameter is zero, the return value is zero.

AAttributes [in, optional]
Set of PAGE_XYZ flags, which indicate access to check.

Optional. Default is to check for any read access (0 means `PAGE_READONLY or PAGE_READWRITE or PAGE_EXECUTE_READ or PAGE_EXECUTE_READWRITE or PAGE_WRITECOPY or PAGE_EXECUTE_WRITECOPY`).

Return value

If the caller has the specified read access to all bytes in the specified memory range, the return value is `ASize`.

If the caller does not have the specified read access to all bytes in the specified memory range, the return value is less than `ASize`. It may be 0.

Remarks

This function is typically used when working with pointers, where you cannot determine the memory management behavior.

Threads in a process are expected to cooperate in such a way that one will not free memory that the other needs. Use of this function does not negate the need to do this. If this is not done, the application may fail in an unpredictable manner.

Dereferencing potentially invalid pointers can disable stack expansion in other threads. A thread exhausting its stack, when stack expansion has been disabled, results in the immediate termination of the parent process, with no pop-up error window or diagnostic information.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

If the calling process has read access to some, but not all, of the bytes in the specified memory range, the return value is in range `0..ASize - 1`.

In a preemptive multitasking environment, it is possible for some other thread to change the process's access to the memory being tested. Even when the function indicates that the process has read access to the specified memory, you should use structured exception handling when attempting to access the memory. Use of structured exception handling enables the system to notify the process if an access violation exception occurs, giving the process an opportunity to handle the exception.

Note
By passing appropriate flags, this function can be used to verify write access to memory.

Note
This function is based on VirtualQuery function. Thus, it doesn't have some of the bad effects of IsBadReadPtr function. However, other caveats still appy.

Warning
Use this routine with caution. It's heuristic. Even when the function indicates that the process has read access to the specified memory, you should use structured exception handling when attempting to access the memory - because other thread may release this memory between checking and actual access.

See also

- IsValidBlockAddr 465

10.130 GetTaskBarWnd

Returns window of application, which creates a taskbar button (if applicable).

Unit

EAppType 54

Syntax

Code (Delphi)

```
function GetTaskBarWnd: HWND;
```

Return value

Handle of application's taskbar window.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

By default it's 0.

Plugin can specify a way to get handle by installing a _GetTaskBarWnd handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note

Typically this is Application.MainForm.Handle for modern applications and Application.Handle for old applications.

See also

- GetMainWnd 424
- GetApplicationWnd 406
- GetParentWnd 426
- _GetTaskBarWnd
- IsAppType 442

10.131 GetTracer

Get stack tracing class by number.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
function GetTracer(
   const ATracerNo: Integer
): TEurekaBaseStackList 172;
```

Parameters

ATracerNo [in]

Number of stack tracer. Numbers are listed as Tracer*XYZ* constants in EStackTracing 133 unit. Some examples are: TracerRawEurekaLogV7 TracerRawEurekaLogV6 TracerRawEurekaLog TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesEurekaLogV7 TracerFramesRecovered TracerFramesEurekaLog TracerFramesJCL TracerLeaksFrame TracerWindows Any other Tracer*XYZ* value...

Return value

Initialized stack tracing class instance (object). Object is created and initialized, but call stack is not build yet.

Remarks

Use this function to get call stack class for specific tracer of your choice.

Tip

Use EurekaCallStackClass 755.`Create` when you don't know which class do you need.

Tip

This function is preferred over GetTracerClass 431 function.

Examples

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
var
   CallStack: TEurekaBaseStackList;
begin
   CallStack := GetTracer(TracerWindows);
   try
      // Build current call stack including current execution point
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- GetTracerClass 431
- TEurekaBaseStackList.GetCurrentInstruction method 173

10.132 GetTracerClass

Get stack tracing class by number.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
function GetTracerClass(
   const ATracerNo: Integer
): TEurekaCallStackClass 678;
```

Parameters

ATracerNo [in]

Number of stack tracer. Numbers are listed as Tracer*XYZ* constants in EStackTracing 133 unit. Some examples are: TracerRawEurekaLogV7 TracerRawEurekaLogV6 TracerRawEurekaLog TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesEurekaLogV7 TracerFramesRecovered TracerFramesEurekaLog TracerFramesJCL TracerLeaksFrame TracerWindows Any other Tracer*XYZ* value...

Return value

Stack tracing class. You can create object by using `Create` constructor.

Remarks

Use this function to get call stack class for specific tracer of your choice.

Tip

Use EurekaCallStackClass 755 when you don't know which class do you need.

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Use GetTracer 430 function instead (when possible).

See also

- GetTracer 430

10.133 GetUserEMail

Gets user's e-mail.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function GetUserEMail: String;
```

Return value

E-mail or empty string.

Example value: 'example@example.com'

Remarks

The e-mail value is stored in the registry under EurekaLog info key. It can be set by the SetUserEMail 553 function. Usually this value is set by dialogs when user enters his e-mail for sending.

See also

- SetUserEMail 553
- GetUserFullName
- GetUserName
- GetComputerName 409
- GetCompanyName 407

10.134 GUIFont

Returns font for GUI applications.

Unit

EFonts 95

Syntax

Code (Delphi)

```
function GUIFont: TFont;
```

Return value

Variable-pitch (proportional space) system font.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function returns font from SystemParametersInfo(SPI_GETNONCLIENTMETRICS).lfMessageFont.

See also

- What are SYSTEM_FONT and DEFAULT_GUI_FONT?
- MonoFont 495

10.135 HandleException

Handles exception.

Unit

EBase 57

Syntax

Code (Delphi)

```
function HandleException(
   const E: TObject = nil;
   const AExpected: Boolean = False;
   const AHandler: String = '';
   const AExceptionAddress: Pointer = nil
): Integer;
```

Parameters

E [in, optional]

Exception object to process. Default is `nil`. If this parameter is not specified, the return value of ExceptObject will be used. Does nothing for EAbort.

Return value

Error code for the E argument. You can use this code to specify failure reason for thread, process, etc. This code is never equals 0, unless there is no exception.

Remarks

Handles (processes) the specified exception. It is usually means saving bug report, displaying EurekaLog's exception dialog, sending bug report, etc.

Note

If EurekaLog is installed - this function will redirect to ExceptionManager.Handle 234.

If EurekaLog is not installed - this function will show simple message box by using ApplicationShowException function (if it is not installed - then `HandleException` will show its own message box for Exception child classes and use ShowException for non-Exception classes).

See also

- ShowExpected Function 557
- TExceptionManager Class 232
- ExceptionManager Function 396
- ExceptionManager.Handle Method 234

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.136 HookWin32API

Adds a function to resource monitoring.

Unit

EResLeaks 116

Syntax

Code (Delphi)

```
function HookWin32API(
   const AModuleName, AAPIName, AAPICategory, AAPIType: String;
   AParamCount: Integer;
   ACreate: Boolean;
   ACheckType: TCheckType 619 = ctEqual;
   ACheckValue: PtrUInt 609 = 0;
   AResultParam: Integer = -1
): Pointer;
```

Parameters

AModuleName
DLL file name to hook. Case insensitive

AAPIName
DLL's function name to hook. Case sensitive

AAPICategory
Category for API.

AAPIType
Sub-type of API. Typically equals APICategory

AParamCount
Number of functions arguments

ACreate
True - function creates resource, False - function destroys resource

ACheckType
Specifies valid/invalid conditions for function call. See TCheckType description for more details

ACheckValue
Value for failure call. See TCheckType description for more details

AResultParam
Index of function's argument which returns resource (for Create-like functions) or accepts resource (for Destroy-like functions). Use -1 for functions and >= 0 for procedures. Use -2 for resources without handles at all (i.e. handle is never exposed, so you only need to watch for pairing calls)

Remarks

Notes:

- ACheckValue is ignored (and must be 0) for ctNone, ctHResult, ctNTStatus.

- AResultParam can't be a return value, if check type is ctHResult or ctNTStatus

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Dispose-like function (ACreate = False) can't return handle - handle should be only passed as argument or be an internal handle (AResultParam = -2 or AResultParam >= 0)

10.137 IfElse

A replacement for ternary `if` operator.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IfElse(
   const AOK: Boolean;
   const AStr1: String;
   const AStr2: String = ''
): String;
```

Parameters

AOK [in]
Condition.

AStr1 [in]
Result for `True`.

AStr2 [in, optional]
Result for `False`.

Return value

Returns `AStr1` if `AOK` is `True`.

Returns `AStr2` if `AOK` is `False`.

Remarks

Warning

Both arguments (AStr1 and AStr2) are calculated. This is different from the real ternary if operator like `AOK ? AStr1 : AStr2` in C++.

10.138 IniReadString

Reads string from .ini file.

Unit

EConfig 63

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function IniReadString(
   const AFileName: String;
   const ASection, AName: String;
   const ADefault: String = ''
): String; deprecated'Consider using ConfigWriteXXX/ConfigReadXXX routines
```

Parameters

AFileName
A full absolute path to file name.

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

ADefault
Default value, if key or section doesn't exits. Optional.

Return value

Returns key's value on success or `ADefault` in case of non-existent key or section.

Returned value needs to be decoded by DecodeConfigValue 352.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

See also

- IniWriteString 440
- ConfigReadString 338

10.139 InitCheckFreeze

Starts checking for freezes and hangs.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
procedure InitCheckFreeze(
   const AThreadID: Cardinal = 0;
   const AMainWnd: HWND = 0;
   const AThreadHandle: THandle = 0;
   const ATimeout: Cardinal = 4;
   const AAntiFreezeMsg: String = '';
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const AAntiFreezeWCTMsg: String = ''
);
```

Parameters

AThreadID [in, optional]
Thread ID to detect hangs inside this thread. Usually this is MainThreadID. Default is 0 - which means "use MainThreadID". This value can be ignored by some checks. For example, Wait Chain Traversal (WCT) check detects deadlocks in any EurekaLog-enabled threads - thus it does not need thread ID.

AMainWnd [in, optional]
A handle of main application form. Default is 0 - which means autodetection. This value can be ignored by some checks. For example, Wait Chain Traversal (WCT) check detects deadlocks in any EurekaLog-enabled threads - thus it does not need window handle.

AThreadHandle [in, optional]
A handle of thread identified by `AThreadID`.

ATimeout [in, optional]
A timeout in seconds. Default is 4. It is recommended to pass `CurrentEurekaLogOptions.FreezeTimeout` here.

AAntiFreezeMsg [in, optional]
A message for freeze exception. Default is `'The application seems to be frozen.'`. It is recommended to pass `CurrentEurekaLogOptions.CustomizedExpandedTexts[mtException_AntiFreeze]` here.

AAntiFreezeWCTMsg [in, optional]
Alternative message for freeze exception. Default is `'A deadlock was detected in thread %d.'`. It is recommended to pass `CurrentEurekaLogOptions.CustomizedExpandedTexts[mtException_AntiFreezeWCT]` here.

Remarks

This function creates a background freeze detection thread. Thread will use current EurekaLog options.

The function will use FreezeThreadClass 756 global variable to determinate which class should be used as background freeze detection thread.

Important!

This function should be called from main thread.

Note

This function is called automatically during initialization of EurekaLog. You can call it manually to re-create/re-initialize freeze detection checks. For example, if you want to change timeout at run-time.

The function will not be called automatically if hang detection is disabled in EurekaLog project options.

Warning

This function do nothing if freeze detection thread is already running. Use DoneCheckFreeze 357 to stop background freeze detection thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note

This function will do nothing if your current settings blocks freeze detection. For example, if your settings indicate that freeze detection should not be active during debugging, and you currently run application under debugging - then this call will do nothing.

See also

- DoneCheckFreeze Function 357
- IsFreezeCheckEnabled Function 457
- PauseFreezeCheck Function 497
- TFreezeThread Class 246
- FreezeThreadClass Variable 756

10.140InitSalt

Generates random salt.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function InitSalt: TSalt 721;
```

Return value

Random salt.

You do not need to free/release/dispose this value - it does not contain any allocated memory. However, it is strongly recommended to call SecureFree 545 function once you have finished working with salt - to erase it with zeros.

This value is never the same.

Remarks

This function generates new random salt for TEADeriveKey 565 function and TwofishDeriveKey 580 function.

Important!

Do not use RTL's Random function to create random salt manually. RTL implementation is a simple pseudo-random generator and it does not generate cryptographically-strong random data.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Examples

Code (Delphi)

```
var
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   Key: TTEAKey;
   Salt: TSalt;
   EncryptedText: RawByteString;
   S: String;
   SaltEncodedLen: Integer;
begin
   // Create new (random) salt
   Salt := InitSalt;

   // Create new key from password
   // Key will be different each time
   // Because salt is different
   Key := TEADeriveKey('password', @Salt);

   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // You must store salt with encrypted data
   EncryptedText :=
      RAWToString(@Salt, SizeOf(Salt)) +
      EncryptedText;

   // No longer needed, erase it
   SecureFree(Salt);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will always be different
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Read stored salt
   SaltEncodedLen := HexCalcEncodedSize(SizeOf(Salt)));
   RAWFromString(Copy(EncryptedText, 1, SaltEncodedLen), @Salt);
   // Remove salt from EncryptedText
   EncryptedText := Copy(EncryptedText, SaltEncodedLen + 1, MaxInt);

   // Create key from password
   // Key will be the same as the one used for encryption above
   Key := TEADeriveKey('password', @Salt);

   // No longer needed, erase it
   SecureFree(Salt);

   // Decrypt text back
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

See also

- TEADeriveKey Function 565
- TwofishDeriveKey Function 580

10.141 InitWER

Initializes EWER unit.

Unit

EWER 140

Syntax

Code (Delphi)

```
procedure InitWER;
```

Remarks

This function imports dynamically linked WER functions (such as ReportFault, WerReportSubmit, etc.).

Call this function before using EWER unit.

10.142 IniWriteString

Writes string to .ini file.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure IniWriteString(
   const AFileName: String;
   const ASection, AName, AValue: String
); deprecated'Consider using ConfigWriteXXX/ConfigReadXXX routines';
```

Parameters

AFileName

A full absolute path to file name. If file doesn't exist - it'll be created (as unicode .ini file).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ASection
Section name. Can't be empty. Can't contain '\' or '/'.

AName
Key name. Can't be empty.

AValue
A value to write. Mandatory, but can be an empty string. Writing empty string will delete the value.

This value needs to be encoded by EncodeConfigValue 384, if you want to preserve special characters.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

See also

- IniReadString 435
- ConfigWriteString 342

10.143 InstructionToString

Converts decoded CPU instruction to string represenstation.

Unit

EDisAsm 78

Syntax

Code (Delphi)

```
function InstructionToString(
   const AInstr: TInstruction 687;
   const AEncrypt: Boolean = False
): String;
```

Parameters

AInstr [in]
Decoded CPU instruction to convert. Usually this value comes from DecodeInstruction function. Exact structure of `TInstruction` depends on current CPU. Please, refer to source code of the unit for your CPU (such as EDisAsmX8632.pas, EDisAsmX8664.pas, etc.).

Return value

Textual representation of `AInstr`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function converts already decoded CPU instruction to string.

Sample output:

```
006989A2 E805C2F8FF CALL -$073DFB ; ($00624BB0) Unit1.TForm1.CheckVisibl
```

- First part ("006989A2" in the above example) repeats `Address` and it's always 8 hex characters for 32-bit process and 16 hex characters for 64-bit process.
- Second part ("E805C2F8FF" in the above example) indicate RAW instruction bytes. Its size varies from 2 characters (1 byte) up to 30 characters (15 bytes) for x86-32 - depending on actual CPU instruction. These bytes includes any prefix, opcode, MOD/RM byte, SIB byte, displacement bytes, immediate value bytes.
- Third part ("CALL -$073DFB" in the above example) shows actual textual representation of disassembled instruction.
- Fourth part ("; ($00624BB0) Unit1.TForm1.CheckVisible" in the above example; optional) contains an optional comment for the instruction (such as call/jmp destination or string data dump).

See also

- DisasmInstruction function 356
- DecodeInstruction function 353
- DisasmFunction function 355

10.144 IntoIDE

Detects if we're running inside IDE.

Unit

ELowLevel 105

Syntax

```
function IntoIDE(
   const AVirtualize: Boolean = True
): Boolean;
```

Return value

True if current process is development IDE (Delphi, C++ Builder and so on), False - otherwise.

Remarks

This function is used to disable EurekaLog functionality in IDE expert.

The check is based on .exe file name.

10.145 IsAppType

Checks if application is of specified type.

Unit

EAppType 54

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function IsAppType(
   const AAppType: TAppType
): Boolean;
```

Parameters

AAppType

Type of application to test. Usually this is one of `ETypes.at`*XXX* constants - like `atVCL`, `atConsole` and so on.

Return value

`True` if some code (plugin) has marked application as being of `AAppType` type. `False` - otherwise.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

Application may have more than one type in some cases. For example, it can be both console and CGI.

Application type is generic qualifier that affects EurekaLog behaviour. For example, VCL application intercepts Application.ShowException and uses window error dialog.

Code can call this routine to check if certain support plugin was installed. For example, `EAppVCL` calls this as `SetAppType(atVCL)` - indicating that application is VCL application.

Plugins can use SetAppType procedure to indicate that they are installed.

There are also simple wrappers for more easy access. For example, IsVCL, IsConsole and so on. See EAppType unit for more info.

See also

- SetAppType

10.146 IsBPL

Checks if specified module is BPL file.

Unit

EAppType

Syntax

Code (Delphi)

```
function IsBPL(
   const AInstance: HMODULE = 0
): Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AInstance

Module handle to analyze. Optional. Default (0) means current module.

Return value

Returns True if `AInstance` module is BPL file. False - otherwise or on error.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- GetModuleType 425
- IsDLL 449
- IsModuleLibrary 462

10.147 IsCBuilder

Indicates if current project was compiled by C++ Builder.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsCBuilder: Boolean;
```

Return value

`True` if project was compiled by C++ Builder; `False` otherwise (e.g. compiled by Delphi).

Remarks

This function is used to determinate if you are running C++ Builder project or Delphi project. This is useful if you need to take C++ Builder specifics into account.

Note
Note that this is not the same as compiler for units. For example, the unit can be compiled by Delphi, but be included/used in C++ Builder project. This function will return `True` for such case, while any compile-time check inside the unit will return `False`.

See also

- IsDynamicRTL 450
- IsLLVM 460

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.148 IsCGI

Checks if application is CGI application.

Unit
EAppType 54

Syntax

Code (Delphi)

```
function IsCGI: Boolean;
```

Return value
Returns True if application is CGI application of any type. False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also
- IsAppType 442
- IsCGIWin 446
- IsCGIConsole 445

10.149 IsCGIConsole

Checks if application is CGI console application.

Unit
EAppType 54

Syntax

Code (Delphi)

```
function IsCGIConsole: Boolean;
```

Return value
Returns True if application is CGI console application (not includes CGI Windows application). False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This is a wrapper for IsAppType 442 function.

Returns "`IsCGI and IsConsole`".

If this function returns True, then IsCGI 445 also returns True. Opposite is not correct.

See also

- IsAppType 442
- IsCGI 445
- IsConsole 447
- IsCGIWin 446

10.150 IsCGIWin

Checks if application is CGI Windows application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsCGIWin: Boolean;
```

Return value

Returns True if application is CGI Windows application (not includes CGI console application). False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

Returns "`IsCGI and (not IsConsole)`".

If this function returns True, then IsCGI 445 also returns True. Opposite is not correct.

See also

- IsAppType 442
- IsCGI 445
- IsConsole 447
- IsCGIConsole 445

10.151 IsCLX

Checks if application is CLX application.

Unit

EAppType 54

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
function IsCLX: Boolean;
```

Return value

Returns True if application is CLX application. False - otherwise or on error.

Remarks

> **Tip**
>
> This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

CLX application is Qt cross-platform application in Delphi/C++ Builder 6 and 7.

See also

- IsAppType 442
- IsVCL 469
- IsFMX 456
- IsCPL 446

10.152 IsCompiledWithPackages

Checks if this module is compiled with packages.

Unit

ELowLevel 105

Syntax

```
function IsCompiledWithPackages: Boolean;
```

Return value

True for packaged module, False - otherwise.

Remarks

This function is used to switch EurekaLog's behaviour between packaged and normal mode.

10.153 IsConsole

Checks if application is console application.

Unit

EAppType 54

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
function IsConsole: Boolean;
```

Return value

Returns True if application is console application. False - otherwise or on error.

Remarks

> Tip
>
> This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- IsAppType 442
- IsCGIConsole 445

10.154 IsCPL

Checks if application is control panel application.

Unit

EAppType 54

Syntax

```
function IsCPL: Boolean;
```

Return value

Returns True if application is CPL application. False - otherwise or on error.

Remarks

> Tip
>
> This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

CPL application is a DLL applet for Control Panel with .cpl extension.

See also

- IsAppType 442
- IsVCL 469
- IsFMX 469
- IsCLX 446

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.155 IsDebugged

Determines whether the calling process is being debugged by a user-mode debugger.

Unit
ELowLevel 105

Syntax

Code (Delphi)

```
function IsDebugged: Boolean;
```

Return value
If the current process is running in the context of a debugger, the return value is `True`.

If the current process is not running in the context of a debugger, the return value is `False`.

Remarks
This function allows an application to determine whether or not it is being debugged, so that it can modify its behavior.

For example, an application could provide additional information using the OutputDebugString function if it is being debugged or execute a `int 3` breakpoint at critical errors (see also DebugBreak 352 function).

See also
- DebugBreak 352

10.156 IsDLL

Checks if specified module is DLL file.

Unit
EAppType 54

Syntax

Code (Delphi)

```
function IsDLL(
   const AInstance: HMODULE = 0
): Boolean;
```

Parameters
`AInstance` [in, optional]
Module handle to analyze. Default (0) means current module.

Return value
Returns True if `AInstance` module is DLL file, but not BPL. False - otherwise or on error.

Remarks

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- GetModuleType 425
- IsBPL 443
- IsModuleLibrary 462

10.157 IsDynamicRTL

Indicates if current project was compiled with dynamic RTL DLL.

Unit

ELowLevel 105

Syntax

Code (Delphi)
`function IsDynamicRTL: Boolean;`

Return value

`True` if project was compiled by C++ Builder with dynamic RTL DLL; `False` otherwise (e.g. compiled by Delphi or without dynamic RTL).

Remarks

This function is used to determinate if you was compiled with dynamicr RTL DLL or not.

Note
This value is always `False` for Delphi projects. Only C++ Builder projects can be compiled with RTL DLL.

See also

- IsCBuilder 444
- IsLLVM 460

10.158 IsEurekaLogActive

Indicates if EurekaLog is active or disabled in this application.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)
`function IsEurekaLogActive: Boolean;`

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

True if EurekaLog is active, False - otherwise.

Remarks

EurekaLog is considered to be active if EurekaLog should do its work. You can control EurekaLog activation by calling SetEurekaLogState 6 procedure.

If EurekaLog was not installed - then it's always disabled (not active).

This is a global status for all application. If you want to check per-thread activation status - use IsEurekaLogActiveInThread 5 function.

If you want to change activation status (enable or disable EurekaLog) - use SetEurekaLogState 6 for global status and SetEurekaLogStateInThread 7 for per-thread status.

By default, EurekaLog is active at startup (if it was installed).

Caution!

This function doesn't respect per-thread activation status. Thus, if EurekaLog was disabled in one thread only - this function will still returns `True`.

This function respects "installed" status, so there is no need for additional call to IsEurekaLogInstalled function.

Note

This function is for the clients only. EurekaLog's code should use _IsEurekaLogActive instead.

Note

This is a wrapper for IsEurekaLogActive 452 EBase.IsEurekaLogActive

Examples

Code (Delphi)

```
if IsEurekaLogActive then
   // this exception will be handled by EurekaLog, since EurekaLog is er
   raise Exception.Create('Error message')
else
   // this exception will not be handled by EurekaLog, since EurekaLog i
   raise Exception.Create('Error message');
```

See also

- IsEurekaLogActiveInThread 5
- IsEurekaLogInstalled 2
- SetEurekaLogState 6
- SetEurekaLogStateInThread 7
- _IsEurekaLogActive
- _InstallEurekaLog

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.159 IsEurekaLogActive

Indicates if EurekaLog is active or disabled in this application.

Unit

EBase 57

Syntax

Code (Delphi)

```
function IsEurekaLogActive: Boolean;
```

Return value

`True` if EurekaLog is active, `False` - otherwise.

Remarks

EurekaLog is considered to be active if EurekaLog should do its work. You can control EurekaLog activation by calling SetEurekaLogState 549 procedure.

If EurekaLog was not installed - then it's always disabled (not active).

This is a global status for all application. If you want to check per-thread activation status - use IsEurekaLogActiveInThread 5 function.

If you want to change activation status (enable or disable EurekaLog) - use SetEurekaLogState 549 for global status and SetEurekaLogStateInThread 7 for per-thread status.

By default, EurekaLog is active at startup (if was installed).

Caution!

This function doesn't respect per-thread activation status. Thus, if EurekaLog was disabled in one thread only - this function will still returns `True`.

This function respects "installed" status, so there is no need for additional call to IsEurekaLogInstalled function.

Note

This function is for the clients only. EurekaLog's code should use _IsEurekaLogActive_IsEurekaLogActive

Examples

Code (Delphi)

```
if IsEurekaLogActive then
   // this exception will be handled by EurekaLog, since EurekaLog is er
   raise Exception.Create('Error message')
else
   // this exception will not be handled by EurekaLog, since EurekaLog i
   raise Exception.Create('Error message');
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- IsEurekaLogInstalled
- IsEurekaLogInstalledInThread
- SetEurekaLogState 549
- SetEurekaLogStateInThread 7
- _IsEurekaLogActive
- _InstallEurekaLog

10.160 IsEurekaLogActiveInThread

Indicates if EurekaLog is active or disabled in the specified thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function IsEurekaLogActiveInThread(
   const AThreadID: Cardinal = 0
): Boolean;
```

Parameters

AThreadID [in, optional]

TID - thread ID which EurekaLog's status should be checked. Use 0 (default) for current thread.

Return value

True if EurekaLog is active in the specified thread, False - otherwise.

Remarks

This is per-thread version of IsEurekaLogActive 3 function.

This is a per-thread status. If you want to check global activation status - use IsEurekaLogActive 3 function.

If you want to change activation status (enable or disable EurekaLog) - use SetEurekaLogState 6 for global status and SetEurekaLogStateInThread 7 for per-thread status.

Caution!

This function doesn't respect global activation status.

Note

This is a wrapper for EThreadsManager.IsEurekaLogActiveInThread function.

Examples

Code (Delphi)

```
if IsEurekaLogActiveInThread then
   // this exception will be handled by EurekaLog, since EurekaLog is er
   raise Exception.Create('Error message')
else
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   // this exception will not be handled by EurekaLog, since EurekaLog i
   raise Exception.Create('Error message');
```

See also

- IsEurekaLogActive 3
- IsEurekaLogInstalled 2
- SetEurekaLogState 6
- SetEurekaLogStateInThread 7
- _InstallEurekaLog
- _IsEurekaLogActive

10.161 IsEurekaLogInstalled

Indicates if EurekaLog is installed in this application.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function IsEurekaLogInstalled: Boolean;
```

Return value

True if EurekaLog is installed, False - otherwise

Remarks

EurekaLog is considered to be installed if:
1. its source code was included into application (see ExceptionLog7 unit 141);
2. executable was processed by EurekaLog expert or command-line compiler (ECC32/EMAKE);
I.e. EurekaLog is installed if both true: source code is included **AND** options is included.

Debug information (which is required to build human-readable call stack) can be included or not - this doesn't matter for installation. EurekaLog can function without debug information.

Alternative to this function can be using of conditional directives.

EurekaLog defines the following conditional symbols:
- EUREKALOG
- EUREKALOG7

Note

This is a wrapper for EBase.IsEurekaLogInstalled function.

Examples

Code (Delphi)

```
uses
   EBase;
// ...
if IsEurekaLogInstalled then
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
begin
   // Do something if EurekaLog is installed
end;
```

Code (Delphi)

```
{$IFDEF EUREKALOG}
   // Do something if EurekaLog is installed
{$ENDIF}
```

See also

- IsEurekaLogActive 3
- _InstallEurekaLog

10.162 IsEXE

Checks if specified module is EXE file.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsEXE(
   const AInstance: HMODULE = 0
): Boolean;
```

Parameters

AInstance [in, optional]
Module handle to analyze. Default (0) means current module.

Return value

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

Remarks

This is a wrapper for GetModuleType 425 function.

See also

- GetModuleType 425

10.163 IsExpected

Checks if exception is an expected exception.

Unit

EBase 57

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function IsExpected(
   const AException: TObject
): Boolean; overload;

function IsExpected(
   const AException: TObject;
   out AContextNumber: Integer;
   out AURL: String;
   out ABugID: TBugID
): Boolean; overload;
```

Parameters

AException [in]
Exception object to check.

AContextNumber [out, optional]
Help context number for this exception.

AURL [out, optional]
URL to the web-page for this exception.

ABugID [out, optional]
BugID for this exception.

Return value

`True` if `AException` is an expected exception, `False` otherwise.

Remarks

Exception can be marked as "expected" by various ways. This function checks if the exception was marked as an expected exception.

See also

- RaiseExpected Function 499
- RaiseOuterExceptionExpected Function 501
- MarkExpected Function 480
- ShowExpected Function 557

10.164 IsFMX

Checks if application is FMX application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsFMX: Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Returns True if application is FMX application. False - otherwise or on error.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

FMX is FireMonkey/FM cross-platform application in Delphi/C++ Builder XE2+.

See also

- IsAppType 442
- IsCLX 446
- IsVCL 469
- IsCPL 446

10.165 IsFreezeCheckEnabled

Indicate if freeze detection is currently active.

Unit

EFreeze 96

Syntax

```
function IsFreezeCheckEnabled: Boolean;
```

Return value

True if freeze detection is working, False otherwise (freeze detection is not enabled or paused).

Remarks

This function indicate if freeze detection was paused by a call of PauseFreezeCheck 497/ResumeFreezeCheck 524 (`eFreezeCheckEFreeze.PauseFreezeCheck`) guarded with `try/except` block.

Note
`IsFreezeCheckEnabled` can be called from any thread.

See also

- PauseFreezeCheck Function 497
- ResumeFreezeCheck Function 524
- IsFreezeCheckInstalled Function 458

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.166 IsFreezeCheckInstalled

Indicate if freeze detection was initialized.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
function IsFreezeCheckInstalled: Boolean;
```

Return value

`True` if freeze detection is installed, `False` otherwise.

Remarks

This function indicate if freeze detection was installed by a call of PauseFreezeCheck 436 function.

Note

We do not recommend to use this function, because freeze checks state may be changed by another thread right after call to this function. We recommend to use paired calls (DoneCheckFreeze 357/InitCheckFreeze 436`heckFreezeEFreeze.DoneCheckFreeze`) when you want to reinitialize freeze detection.

Note

`IsFreezeCheckInstalled` can be called from any thread.

See also

- InitCheckFreeze Function 436
- DoneCheckFreeze Function 357
- IsFreezeCheckEnabled Function 457

10.167 IsGUI

Checks if application is GUI application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsGUI: Boolean;
```

Return value

Returns True if application is GUI application. False - otherwise or on error.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

GUI application is VCL, CLX, VCLX or CPL application.

Returns "`IsVCL or IsCLX or IsFMX or IsCPL`".

See also

- IsAppType 442
- IsVCL 469
- IsCLX 446
- IsFMX 456
- IsCPL 448
- IsConsole 447
- IsService 464

10.168 IsIntraWeb

Checks if application is IntraWeb application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsIntraWeb: Boolean;
```

Return value

Returns True if application is IntraWeb application. False - otherwise or on error.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- IsAppType 442

10.169 IsISAPI

Checks if application is ISAPI application.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsISAPI: Boolean;
```

Return value

Returns True if application is ISAPI application. False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- IsAppType 442

10.170 IsLLVM

Indicates if current project is compiled by "new" C++ Builder compiler (e.g. LLVM).

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsLLVM: Boolean;
```

Return value

`True` if project was compiled by LLVM Builder; `False` otherwise (e.g. compiled by Delphi or classic C++ Builder).

Remarks

This function is used to determinate if you are running under LLVM. This is useful if you need to take LLVM specifics into account.

Note

Note that this is not the same as compiler for units. For example, the unit can be compiled by Delphi, but be included/used in C++ Builder project. This function will return `True` for such case, while any compile-time check inside the unit will return `False`.

See also

- IsCBuilder 444

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- IsDynamicRTL 450

10.171 IsMemLeaksEnabled

Indicate if memory leaks checks are enabled.

Unit
EMemLeaks 109

Syntax

Code (Delphi)

```
function IsMemLeaksEnabled: Boolean;
```

Return value
`True` - memory leaks checks are enabled, `False` otherwise.

Remarks
This function indicate if memory leaks checks are enabled.
- When enabled, EurekaLog will collect call stack information of each allocated memory block. Additional information (such as name, grouping, etc.) will also be stored.
- When disabled, EurekaLog will not collect call stack and additional information. Overhead will be significantly reduced.

See also description of EMemLeaks unit 109.

This is a global state, which can be changed via EnableMemLeaksRegistering 383/ DisableMemLeaksRegistering 354 or SetMemLeaksRegistering functions. You may temporally disable memory leaks checks when executing code with large amount of memory allocations which are guaranteed not to leak.

Note

This function will always return `False` if EurekaLog is not installed. See IsMemLeaksInstalled function 461.

Tip

This function also returns `False` when inside EurekaLog's own code.

See also
- EnableMemLeaksRegistering Function 383
- DisableMemLeaksRegistering Function 354
- SetMemLeaksRegistering Function
- IsMemLeaksInstalled Function 461

10.172 IsMemLeaksInstalled

Indicate if EurekaLog's memory filter is installed.

Unit
EMemLeaks 109

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
function IsMemLeaksInstalled: Boolean;
```

Return value

`True` if filter is installed, `False` otherwise.

Remarks

This value indicate if EurekaLog has installed memory filter to add additional functionality (memory problems checks).

When this value is `True` - EurekaLog's filter is used, overhead is added. When this value is `False` - native memory manager is used, no overhead. See also description of EMemLeaks unit 109.

Note

This value is initialized on startup and never changes.

See also

- IsMemLeaksEnabled Function 461

10.173 IsModuleLibrary

Checks if specified module is DLL or BPL file.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsModuleLibrary(
   const AInstance: HMODULE = 0
): Boolean;
```

Parameters

`AInstance`

Module handle to analyze. Optional. Default (0) means current module.

Return value

Returns True if `AInstance` module is DLL file (it could be a BPL too). False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

See also

- GetModuleType 425

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- IsDLL 449
- IsBPL 443

10.174 IsReadableFile

Verifies that the caller has read access to the specified file.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsReadableFile(
   const AFileName: String
): Boolean;
```

Parameters

AFileName [in]
A full absolute file name.

Return value

If the current process have a GENERIC_READ access to AFileName file, the return value is True.

If the current process doesn't have a GENERIC_READ access to AFileName file, the return value is False.

Remarks

This function tries to open specified file with GENERIC_READ access. If this operation succeeds - the function returns True, otherwise - False.

Thus, function returns False, if file doesn't exist at all.

See also

- IsWritableFile 477

10.175 IsRTLCompiledWithStackFrames

Determines whether the RTL was compiled with Stack Frames option enabled.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsRTLCompiledWithStackFrames: Boolean;
```

Return value

If RTL was compiled with Stack Frames option, the return value is True.

If RTL was compiled without Stack Frames option, the return value is False.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This is a simple wrapper for IsStackFramesProc 464 with arbitrary simple function from RTL.

Warning
This is heuristic function.

See also

- IsStackFramesProc 464

10.176 IsService

Checks if application is a win32 service application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsService: Boolean;
```

Return value

Returns True if application is a service. False - otherwise or on error.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

Service application can contain one or more Win32 services.

See also

- IsAppType 442
- IsVCL 469
- IsConsole 447

10.177 IsStackFramesProc

Determines whether the specified function has a stack frame.

Unit

ELowLevel 105

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function IsStackFramesProc(
   const AAddr: Pointer
): Boolean;
```

Parameters

AAddr [in]
A pointer to function.

Return value

If specified function has a stack frame, the return value is `True`.

If specified function doesn't have a stack frame, the return value is `False`.

Remarks

Note

Make sure that you pass a pointer to a real function. Not to some arbitrary code or function's stub.

Use to get pointer to real code from stub.

Warning

This is heuristic function.

See also

- IsRTLCompiledWithStackFrames 463

10.178 IsValidBlockAddr

Verifies that the caller has read access to the specified range of memory.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsValidBlockAddr(
   const Address: Pointer;
   const Size: Cardinal
): Boolean;
```

Parameters

Address [in]
A pointer to the first byte of the memory block.

Size [in]
The size of the memory block, in bytes. If this parameter is zero, the return value is `True`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

If the caller has any read access to all bytes in the specified memory range, the return value is `True`.

If the caller does not have a read access to all bytes in the specified memory range, the return value is `False`.

Remarks

This is the simple wrapper for GetReadableSize 428 function.

See GetReadableSize 428 for more information.

Warning

Use this routine with caution. It's heuristic. Even when the function indicates that the process has read access to the specified memory, you should use structured exception handling when attempting to access the memory - because other thread may release this memory between checking and actual access.

See also

- GetReadableSize 428
- IsValidCodeAddr 467
- IsValidClass 466
- IsValidObject 468
- IsValidHandle 468
- IsValidRunningThread
- IsValidSymbolName

10.179 IsValidClass

Verifies that the specified pointer is a valid class.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsValidClass(
   ClassType: TClass
): Boolean;
```

Parameters

`ClassType` [in]
A class pointer.

Return value

If `ClassType` is a real class, the return value is `True`.

If `ClassType` is not a class, the return value is `False`.

Remarks

This function performs many complex checks on passed value to determinate if this is a valid class reference.

Warning

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Use this routine with caution. It's heuristic.

See also
- IsValidBlockAddr 465
- IsValidCodeAddr 467
- IsValidObject 468
- IsValidHandle 468
- IsValidRunningThread
- IsValidSymbolName

10.180 IsValidCodeAddr

Verifies that the caller has execute access to the specified range of memory.

Unit
ELowLevel 105

Syntax

Code (Delphi)

```
function IsValidCodeAddr(
   const AAddress: Pointer;
   const ASize: Cardinal
): Boolean;
```

Parameters
AAddress [in]
A pointer to the first byte of the memory block.

ASize [in]
The size of the memory block, in bytes. If this parameter is zero, the return value is `True`.

Return value
If the caller has any execute access to all bytes in the specified memory range, the return value is `True`.

If the caller does not have a execute access to all bytes in the specified memory range, the return value is `False`.

Remarks
This is the simple wrapper for GetReadableSize 428 function.

See GetReadableSize 428 for more information.

Warning

Use this routine with caution. It's heuristic. Even when the function indicates that the process has execute access to the specified memory, you should use structured exception handling when attempting to access the memory - because other thread may release this memory between checking and actual access.

See also
- GetReadableSize 428
- IsValidBlockAddr 465

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- IsValidClass 466
- IsValidObject 468
- IsValidHandle 468
- IsValidRunningThread
- IsValidSymbolName

10.181 IsValidHandle

Verifies that this is a valid kernel handle.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function IsValidHandle(
   const AHandle: THandle
): Boolean;
```

Parameters

AHandle [in]

A handle to check.

Return value

If the AHandle is a valid kernel handle, the return value is True.

If the AHandle is not a valid kernel handle, the return value is False.

Remarks

This function uses GetHandleInformation and DuplicateHandle to verify handle.

Warning

Use this routine with caution. It's heuristic.

Even when the function indicates that the handle is valid, a handle may be immediately closed by another thread.

See also

- IsValidBlockAddr 465
- IsValidCodeAddr 467
- IsValidClass 466
- IsValidObject 468
- IsValidRunningThread
- IsValidSymbolName

10.182 IsValidObject

Verifies that the specified pointer is a valid object.

Unit

ELowLevel 105

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function IsValidObject(
   const AObj: TObject;
   const AStrict: Boolean
): Boolean;
```

Parameters

AObj [in]
An object pointer.

AStrict [in, optional]
Default is `True`. Indicates if object must be compatible with current language (default). Or if it can be just a valid object from a different IDE/compiler version. For example, if `AStrict` is `True` - then the function returns `False` when called on a valid Delphi 2007 object while being called from Delphi 2009.

Return value

If `AObj` is a real object, the return value is `True`.

If `AObj` is not an object, the return value is `False`.

Remarks

This function performs many complex checks on passed value to determinate if this is a valid object reference.

Warning

Use this routine with caution. It's heuristic.

See also

- IsValidBlockAddr 465
- IsValidCodeAddr 467
- IsValidClass 466
- IsValidHandle 468
- IsValidRunningThread
- IsValidSymbolName

10.183 IsVCL

Checks if application is VCL application.

Unit

EAppType 54

Syntax

Code (Delphi)

```
function IsVCL: Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Returns True if application is VCL application. False - otherwise or on error.

Remarks

Tip
This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

VCL is default usual application in Delphi/C++ Builder for Win32.

See also

- IsAppType 442
- IsCLX 446
- IsFMX 456
- IsCPL 446

10.184 IsVirtualized

Checks if application was instrumented/virtualized.

Unit

ELowLevel 105

Syntax

Code (Delphi)
`function IsVirtualized: Boolean;`

Return value

If the current process is instrumented or virtualized (for example, running by AQTime), the return value is `True`.

If the current process is running as usual, the return value is `False`.

Remarks

This is informational routine.

Warning
EurekaLog may not operate with full features when application is instrumented. That is because code instrumentation introduces dynamically generated code, as well as changes callers of the code - which makes various things impossible to hook.

10.185 IsWeb

Checks if application is web application.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EAppType 54

Syntax

Code (Delphi)

```
function IsWeb: Boolean;
```

Return value
Returns True if application is web application. False - otherwise or on error.

Remarks

Tip

This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

This is a wrapper for IsAppType 442 function.

Web application is CGI, ISAPI or IntraWeb application.

Returns "`IsCGI or IsISAPI or IsIntraWeb`".

See also
- IsAppType 442
- IsCGI 445
- IsISAPI 459
- IsIntraWeb 459

10.186 IsWin10

Detect Windows 10 (minimum).

Unit
ESysInfo 134

Syntax

Code (Delphi)

```
function IsWin10: Boolean;
```

Return value
`True` if current computer is running under Windows 10 **or better**, `False` otherwise.

Remarks
This function will return `True` for both consumer (Windows 10) and server operating system.

This function will return `True` for any later operating systems.

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This function can be used to determinate minimum required level of operating system.

See also

- IsWin11 472
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.187 IsWin11

Detect Windows 11 (minimum).

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function IsWin11: Boolean;
```

Return value

`True` if current computer is running under Windows 11 **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows 11) and server operating system.

This function will return `True` for any later operating systems.

Tip

This function can be used to determinate minimum required level of operating system.

See also

- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.188 IsWin64

Detect OS bitness.

Unit

ESysInfo 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
function IsWin64: Boolean;
```

Return value

`True` if current computer is running under 64-bit Windows, `False` otherwise.

Remarks

This function will return `True` for Windows XP 64-bit, Windows Vista 64-bit, Windows 7 64-bit, their server equivalents or any later 64-bit systems.

Tip

The result of this function doesn't depend on bitness of calling application.

See also

- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWorkstation 476

10.189 IsWin7

Detect Windows 7 (minimum).

Unit

ESysInfo 134

Syntax

```
function IsWin7: Boolean;
```

Return value

`True` if current computer is running under Windows 7 **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows 7) and server (Windows Server 2008 R2) operating system.

This function will return `True` for any later operating systems.

Tip

This function can be used to determinate minimum required level of operating system.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWinVista 475
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.190 IsWin8

Detect Windows 8 (minimum).

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function IsWin8: Boolean;
```

Return value

`True` if current computer is running under Windows 8 **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows 8) and server (Windows Server 2012) operating system.

This function will return `True` for any later operating systems.

Tip

This function can be used to determinate minimum required level of operating system.

See also

- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.191 IsWin81

Detect Windows 8.1 (minimum).

Unit

ESysInfo 134

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
function IsWin81: Boolean;
```

Return value

`True` if current computer is running under Windows 8.1 **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows 8.1) and server (Windows Server 2012 R2) operating system.

This function will return `True` for any later operating systems.

```
Tip
This function can be used to determinate minimum required level of operating system.
```

See also

- IsWin11 472
- IsWin10 471
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.192 IsWinVista

Detect Windows Vista (minimum).

Unit

ESysInfo 134

Syntax

```
Code (Delphi)
function IsWinVista: Boolean;
```

Return value

`True` if current computer is running under Windows Vista **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows Vista) and server (Windows Server 2008) operating system.

This function will return `True` for any later operating systems (Windows 7, etc.).

```
Tip
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This function can be used to determinate minimum required level of operating system.

See also

- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinXP 476
- IsWin64 472
- IsWorkstation 476

10.193 IsWinXP

Detect Windows XP (minimum).

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
function IsWinXP: Boolean;
```

Return value

`True` if current computer is running under Windows XP **or better**, `False` otherwise.

Remarks

This function will return `True` for both consumer (Windows XP) and server (Windows Server 2003) operating system.

This function will return `True` for any later operating systems (Windows Vista, Windows 7, etc.).

If function has returned `False` - the computer is running under Windows 2000.

Tip

This function can be used to determinate minimum required level of operating system.

See also

- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWin64 472
- IsWorkstation 476

10.194 IsWorkstation

Gets server/workstation status of current computer.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ESysInfo 134

Syntax

Code (Delphi)
```
function IsWorkstation: Boolean;
```

Return value
`True` if current computer is workstation, `False` otherwise (it's server).

Remarks
Function determinate server/workstation status based on operating system version. For consumer operating systems (for example - Window XP, Windows 7) it will return `True`. For server operating systems (for example, Windows Server 2003, Windows Server 2008) it will return `False`.

This function can be used to distinguish between, say, Windows 7 and Windows 2008 R2, etc.

See also
- IsWin11 472
- IsWin10 471
- IsWin81 474
- IsWin8 474
- IsWin7 473
- IsWinVista 475
- IsWinXP 476
- IsWin64 472

10.195 IsWritableFile

Verifies that the caller has write access to the specified file.

Unit
ELowLevel 105

Syntax

Code (Delphi)
```
function IsWritableFile(
   const AFileName: String
): Boolean;
```

Parameters
`AFileName` [in]
A full absolute file name.

Return value
If the current process have a `GENERIC_WRITE` access to `AFileName` file, the return value is `True`.

If the current process doesn't have a `GENERIC_WRITE` access to `AFileName` file, the return value is `False`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function tries to open specified file with `GENERIC_WRITE` access. If this operation succeeds - the function returns `True`, otherwise - `False`.

If file doesn't exists, function tries to create it. However, file is deleted after the check.

See also

- IsReadableFile 463

10.196 LastBugID

Returns BugID of last exception in current thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function LastBugID: TBugID 613;
```

Return value

BugID of last thread exception or 0 if there was no exception raised.

See also

- BugID

10.197 LocationToStr

Converts location information to single line string.

Unit

EDebugInfo 68

Syntax

Code (Delphi)

```
function LocationToStr(
   const AInfo: TELLocationInfo 662;
   const AIncludeModuleName: Boolean = True;
   const AIncludeAddressOffset: Boolean = True;
   const AIncludeStartProcLineOffset: Boolean = False;
   const AIncludeVAddress: Boolean = True;
   const AIncludePointer: Boolean = True;
   const AIncludeLine: Boolean = True;
   const AUseParent: TELUseParent 670 = eupNone
): String;
```

Parameters

AInfo [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

A location information to be converted to string. It can be retrieved from GetLocationInfo 423 function.

AIncludeModuleName [in, optional]
Specifies if result should include module name.

AIncludeAddressOffset [in, optional]
Specifies if result should include offset from the routine.

AIncludeStartProcLineOffset [in, optional]
Specifies if result should include offset from the line.

AIncludeVAddress [in, optional]
Specifies if result should include offset from module.

AIncludePointer [in, optional]
Specifies if result should include RAW pointer.

AIncludeLine [in, optional]
Specifies if result should include line number.

AUseParent [in, optional]
Specifies if result should include information about parent or child units (for includes only).

Return value
A human-readable single line string.

Remarks
This function converts location information record into single string, which can be shown to user.

See also
- GetLocationInfo Function 423
- __MODULE__ function
- __UNIT__ function
- __FILE__ function
- __FUNCTION__ function
- __LINE__ function

10.198 MainThreadTopOfStack

Returns top of the stack for the main thread.

Unit
ELowLevel 105

Syntax

Code (Delphi)

```
function MainThreadTopOfStack: Pointer;
```

Return value
Pointer to top of the main thread's stack.

Remarks
Main thread for DLL is a thread which loaded DLL.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.199 MarkExpected

Marks exception as expected exception.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure MarkExpected(
   const AException: TObject;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID = 0
); overload;

procedure MarkExpected(
   const AException: TObject;
   const AURL: String;
   const ABugID: TBugID = 0
); overload;
```

Parameters

AException [in]
Exception object to mark as expected.

AContextNumber [in, optional]
Help context number for this exception.

ABugID [in, optional]
BugID for this exception.

AURL [in, optional]
URL for this exception.

Remarks

Marks exception as "expected". Additionally, sets ExpectedNumber/ ExpectedURL 197 and BugID 187 properties.

See also

- RaiseExpected Function 499
- RaiseOuterExceptionExpected Function 501
- ShowExpected Function 557
- IsExpected Function 455

10.200 MD5Hash

Calculates MD5 hash.

Unit

EHash 97

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function MD5Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TMD5Hash[705]; overload;

function MD5Hash(
   const AData: RawByteString
): TMD5Hash[705]; overload;
```

Parameters

AData [in]
A data to calculate hash.

ADataSize [in]
Size of AData in bytes.

Return value

MD5 hash of the specified AData.

Remarks

The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32 digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications, and is also commonly used to verify data integrity.

Important!

The security of the MD5 hash function is severely compromised. As of 2010, the CMU Software Engineering Institute considers MD5 "cryptographically broken and unsuitable for further use".

Note

This routine is used for generating encryption keys from password for TEA cipher (password-protected debug information).

See also

- MD5HMAC Function[481]

10.201MD5HMAC

Calculates HMAC-MD5.

Unit

EHash[97]

Syntax

Code (Delphi)

```
function MD5HMAC(
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const AData: Pointer;
   const ADataSize: Cardinal;
   const APassword: Pointer;
   const APasswordSize: Cardinal
): TMD5Hash[705]; overload;

function MD5HMAC(
   const AData, APassword: RawByteString
): TMD5Hash[705]; overload;
```

Parameters

AData [in]
A data to calculate value ("text", "message").

ADataSize [in]
Size of AData in bytes.

APassword [in]
A common shared secret ("password", "key").

APasswordSize [in]
Size of APassword in bytes.

Return value

MD5 HMAC value of the specified AData and APassword.

Remarks

A Keyed-Hash Message Authentication Code (HMAC) is a specific type of Message Authentication Code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.

Any cryptographic hash function, such as MD5, SHA-1, or SHA-256, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5, HMAC-SHA1, or HMAC-SHA256 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

Important!
The security of the MD5 hash function is severely compromised. As of 2010, the CMU Software Engineering Institute considers MD5 "cryptographically broken and unsuitable for further use".

See also

- MD5Hash Function[480]
- SHA1HMAC Function[554]
- SHA256HMAC Function[556]

10.202MemLeaksGetName

Returns memory block's name.

Unit

EMemLeaks[109]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function MemLeaksGetName(
   const ABlock: Pointer
): String;
```

Parameters

ABlock [in]
A pointer to memory block.

Return value

Name of the memory block previously set by the MemLeaksName 483 function or an empty string if the name was not set.

Remarks

This function returns a custom name to any memory block allocated with EurekaGetMem with enabled registering leaks. A custom name will be visible in leak reports - which should allow you to identify memory block.

Tip

Name have limited length. Names are stored internally with TCustomName field. Owners of Enterprise version of EurekaLog (with full sources) can alter size of the name and recompile EurekaLog.

Important!

You must pass a pointer to the beginning of the memory block. Please note that dynamic arrays and strings do not point to the beginning of allocation.

See also

- MemLeaksName Function 483

10.203 MemLeaksName

Names a memory block.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksName(
   const ABlock: Pointer;
   const AName: String
);
```

Parameters

ABlock [in]
A pointer to memory block.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AName [in]
A custom name for the ABlock block.

Remarks

This function assigns a custom name to any memory block allocated with EurekaGetMem with enabled registering leaks. A custom name will be visible in leak reports - which should allow you to identify memory block.

Tip

Name have limited length. Names are stored internally with TCustomName field. Owners of Enterprise version of EurekaLog (with full sources) can alter size of the name and recompile EurekaLog.

Important!

You must pass a pointer to the beginning of the memory block. Please note that dynamic arrays and strings do not point to the beginning of allocation. Use MemLeaksNameDynArray 484 and MemLeaksNameStr 485 for dynamic arrays and strings instead of this function.

See also

- MemLeaksOwn Function 485
- MemLeaksNameDynArray Function 484
- MemLeaksNameStr Function 485
- DumpAllocationsToFile Function 358
- MemLeaksGetName Function 482

10.204 MemLeaksNameDynArray

Names a memory block (variant for dynamic arrays).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksNameDynArray(
   const ABlock: Pointer;
   const AName: String
);
```

Parameters

ABlock [in]
A dynamic array casted to pointer.

AName [in]
A custom name for the ABlock block.

Remarks

This function is the same as MemLeaksName function 483, but it accepts only pointers to dynamic arrays.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- MemLeaksName Function 483

10.205 MemLeaksNameStr

Names a memory block (variant for strings).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksNameStr(
   const ABlock: Pointer;
   const AName: String
);
```

Parameters

ABlock [in]

A string casted to pointer.

AName [in]

A custom name for the ABlock block.

Remarks

This function is the same as MemLeaksName function 483, but it accepts only pointers to strings.

See also

- MemLeaksName Function 483

10.206 MemLeaksOwn

Sets ownership for memory block.

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksOwn(
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters

AParentBlock [in]

A pointer to parent block.

AChildBlock [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

A pointer to child block.

AName [in, optional]
A custom name for the AChildBlock block. Empty string (default) means "do not change name".

Remarks

This function is a combination of MemLeaksSetParentBlock 489 and MemLeaksName 483 functions. MemLeaksOwn will name AChildBlock with AName (as MemLeaksName 483 do) and sets AParentBlock as a parent to AChildBlock (as MemLeaksSetParentBlock 489 do).

The difference is that MemLeaksOwn will not overwrite a parent link in AChildBlock if that link was already set. Whereas MemLeaksSetParentBlock 489 will always set parent link. This difference allows you to use MemLeaksOwn to set ownership for reference-counted data types passed to you as arguments. For example, if string is passed to constructor of your object, and you save this string into field of your object - you may use MemLeaksOwnStr 488 to claim ownership of this string.

Important!

You must pass a pointer to the beginning of the memory block. Please note that dynamic arrays and strings do not point to the beginning of allocation. Use MemLeaksOwnDynArrayChild 487, MemLeaksOwnDynArrayParent 487, MemLeaksOwnDynArrayBoth 486 and MemLeaksOwnStr 488 for dynamic arrays and strings instead of this function.

See also

- MemLeaksName Function 483
- MemLeaksSetParentBlock Function 489
- MemLeaksOwnDynArrayChild Function 487
- MemLeaksOwnDynArrayParent Function 487
- MemLeaksOwnDynArrayBoth Function 486
- MemLeaksOwnStr Function 488
- DumpAllocationsToFile Function 358

10.207 MemLeaksOwnDynArrayBoth

Sets ownership for memory block (variant for dynamic arrays).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksOwnDynArrayBoth(
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters

AParentBlock [in]
A dynamic array casted to pointer.

AChildBlock [in]
A dynamic array casted to pointer.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AName [in, optional]
A custom name for the AChildBlock block. Empty string (default) means "do not change name".

Remarks
This function is the same as MemLeaksOwn function 485, but it accepts only pointers to dynamic arrays.

See also
- MemLeaksOwn Function 485

10.208 MemLeaksOwnDynArrayChild

Sets ownership for memory block (variant for dynamic arrays).

Unit
EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksOwnDynArrayChild(
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters
AParentBlock [in]
A pointer to parent block.

AChildBlock [in]
A dynamic array casted to pointer.

AName [in, optional]
A custom name for the AChildBlock block. Empty string (default) means "do not change name".

Remarks
This function is the same as MemLeaksOwn function 485, but it accepts only pointers to dynamic arrays.

See also
- MemLeaksOwn Function 485

10.209 MemLeaksOwnDynArrayParent

Sets ownership for memory block (variant for dynamic arrays).

Unit
EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksOwnDynArrayParent(
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters

AParentBlock [in]
A dynamic array casted to pointer.

AChildBlock [in]
A pointer to parent block.

AName [in, optional]
A custom name for the AChildBlock block. Empty string (default) means "do not change name".

Remarks

This function is the same as MemLeaksOwn function 485, but it accepts only pointers to dynamic arrays.

See also

- MemLeaksOwn Function 485

10.210 MemLeaksOwnStr

Sets ownership for memory block (variant for strings).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksOwnStr(
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters

AParentBlock [in]
A pointer to parent block.

AChildBlock [in]
A string casted to pointer.

AName [in, optional]
A custom name for the AChildBlock block. Empty string (default) means "do not change name".

Remarks

This function is the same as MemLeaksOwn function 485, but it accepts only pointers to strings.

See also

- MemLeaksOwn Function 485

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.211 MemLeaksSetParentBlock

Unit
EMemLeaks 109

Syntax

Code (Delphi)
```
procedure MemLeaksSetParentBlock(
   const AParentBlock, AChildBlock: Pointer
);
```

See also
- MemLeaksOwn Function 485
- MemLeaksSetParentBlockDynArrayChild Function 489
- MemLeaksSetParentBlockDynArrayParent Function 490
- MemLeaksSetParentBlockDynArrayBoth Function 489
- MemLeaksSetParentBlockStr Function 491
- DumpAllocationsToFile Function 358

10.212 MemLeaksSetParentBlockDynArrayBoth

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

Unit
EMemLeaks 109

Syntax

Code (Delphi)
```
procedure MemLeaksSetParentBlockDynArrayBoth(
   const AParentBlock, AChildBlock: Pointer
);
```

Parameters
AParentBlock [in]
A dynamic array casted to pointer.

AChildBlock [in]
A dynamic array casted to pointer.

Remarks
This function is the same as MemLeaksSetParentBlock function 489, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 489

10.213 MemLeaksSetParentBlockDynArrayChild

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EMemLeaks 109

Syntax

Code (Delphi)
```
procedure MemLeaksSetParentBlockDynArrayChild(
   const AParentBlock, AChildBlock: Pointer
);
```

Parameters
AParentBlock [in]
A pointer to parent block.

AChildBlock [in]
A dynamic array casted to pointer.

Remarks
This function is the same as MemLeaksSetParentBlock function 489, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 489

10.214 MemLeaksSetParentBlockDynArrayParent

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

Unit
EMemLeaks 109

Syntax

Code (Delphi)
```
procedure MemLeaksSetParentBlockDynArrayParent(
   const AParentBlock, AChildBlock: Pointer
);
```

Parameters
AParentBlock [in]
A dynamic array casted to pointer.

AChildBlock [in]
A pointer to child block.

Remarks
This function is the same as MemLeaksSetParentBlock function 489, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 489

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.215MemLeaksSetParentBlockStr

Sets parent-child relation between two memory blocks for grouping (variant for strings).

Unit

EMemLeaks 109

Syntax

Code (Delphi)

```
procedure MemLeaksSetParentBlockStr(
   const AParentBlock, AChildBlock: Pointer
);
```

Parameters

AParentBlock [in]
A pointer to parent block.

AChildBlock [in]
A string casted to pointer.

Remarks

This function is the same as MemLeaksSetParentBlock function 489, but it accepts only pointers to strings.

See also

- MemLeaksSetParentBlock Function 489

10.216MemProtect

Encrypts process memory.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure MemProtect(
   const AData: Pointer;
   const ADataSize: Cardinal;
   out AProtectedData: Pointer;
   out AProtectedDataSize: Cardinal
); overload;

function MemProtect(
   const AData: RawByteString
): RawByteString; overload;

procedure MemProtect(
   var AData: TEncryptBuffer 671
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure MemProtect(
   const AData: TEncryptBuffer 671;
   out AProtectedData: TEncryptBuffer 671
); overload;
```

Parameters

AData [in]
Sensitive data to encrypt.

ADataSize [in]
Size of AData.

AProtectedData [out]
Encrypted data. Can be deleted by FreeMem function (but we recommend to use SecureFree 545 function instead).

AProtectedDataSize [out]
Size of AProtectedData.

AData [in]
Sensitive data to encrypt.

AData [in, out]
On input: sensitive data to encrypt. **Must** be allocated with AllocMem function.
On output: encrypted data (size will be different). Can be deleted by EncryptBufferFree 385 or SecureFree 545 functions.

AProtectedData [out]
Encrypted data (size will be different). Delete this data by EncryptBufferFree 385 or SecureFree 545 functions.

Return value

Encrypted data (size will be different).

Remarks

This function encrypts memory to prevent others from viewing sensitive information in your process. For example, use the MemProtect function to encrypt memory that contains a password. Encrypting the password prevents others from viewing it when the process is paged out to the swap file. Otherwise, the password is in plaintext and viewable by others.

Using MemProtect and MemUnprotect for password encryption is not secure because the data exists as plaintext in memory before it is encrypted and at any time the caller decrypts it for use.

Typically, you use the MemProtect function to encrypt sensitive information that you are going to decrypt while your process is running. Do not use this function to save data that you want to decrypt later; you will not be able to decrypt the data if the process or computer is restarted. To save encrypted data to a file to decrypt later, use the DataProtect 349, TEAEncrypt 572, or RSAEncrypt 528 functions.

Call the MemUnprotect 493 function to decrypt memory encrypted with the MemProtect function. When you have finished using the sensitive information, clear it from memory by calling the FillChar function.

Typically, only a user with the same logon credential as the user who encrypted the data can decrypt the data. In addition, the encryption and decryption usually must be done on the same computer. Moreover, decryption usually must be done within the same process.

The function creates a session key to perform the encryption. The session key is derived again when the data is to be decrypted.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

The function also adds a Message Authentication Code (MAC) (keyed integrity check) to the encrypted data to guard against data tampering.

See also

- MemUnprotect Function 493
- DataProtect Function 349
- TEAEncrypt Function 572
- RSAEncrypt Function 528

10.217MemUnprotect

Decrypts process memory.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure MemUnprotect(
   const AProtectedData: Pointer;
   const AProtectedDataSize: Cardinal;
   out AData: Pointer;
   out ADataSize: Cardinal
); overload;

function MemUnprotect(
   const AProtectedData: RawByteString
): RawByteString; overload;

procedure MemUnprotect(
   var AProtectedData: TEncryptBuffer 671
); overload;

procedure MemUnprotect(
   const AProtectedData: TEncryptBuffer 671;
   out AData: TEncryptBuffer 671
); overload;
```

Parameters

AProtectedData [in]
Encrypted data to decrypt.

AProtectedDataSize [in]
Size of AProtectedData.

AData [out]
Decrypted sensitive data. Can be deleted by FreeMem function (but we recommend to use SecureFree 545 function instead).

ADataSize [out]
Size of AData.

AProtectedData [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Encrypted data to decrypt.

AProtectedData [in, out]
On input: encrypted data to decrypt. **Must** be allocated with AllocMem function.
On output: unencrypted sensitive data (size will be different). Can be deleted by EncryptBufferFree 385 or SecureFree 545 functions.

AData [out]
Unencrypted sensitive data (size will be different). Delete this data by EncryptBufferFree 385 or SecureFree 545 functions.

Return value
Unencrypted sensitive data (size will be different).

Remarks
`MemUnprotect` function decrypts memory that was encrypted using the MemProtect 491 function.

Using `MemProtect` and `MemUnprotect` for password encryption is not secure because the data exists as plaintext in memory before it is encrypted and at any time the caller decrypts it for use.

You must encrypt and decrypt the memory during the same boot session. If the computer is restarted before you call the `MemUnprotect` function, you will not be able to decrypt the data.

When you have finished using the sensitive information, clear it from memory by calling the FillChar function.

See also
- MemProtect Function 491
- TEADecrypt Function 559
- RSADecrypt Function 524

10.218 MemWipeStack

Clears unused stack of the current thread.

Unit
ELowLevel 105

Syntax

Code (Delphi)
```
procedure MemWipeStack;
```

Remarks
This function wipes (clears) the content of the unused part of the stack. The wipe is performed with debug pattern ($CCCCCCCC).

This means that when execution goes further - the uninitialized local variables will have non-0, non-nil, unaccessible value. This can help catch use of uninitialized variables.

Additionally, clearing stack can help to reduce number of false-positive entries with RAW stack tracing.

We recommend to call this function from time to time. For example, when application is idle, or about to do some heavy work.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Warning

It is highly recommended to call the function 328 first to determinate if the `MemWipeStack` function should be called.

Caution!

This function clears the entire unused stack to the end. E.g. if you use 1 Mb stack (default) - then 1 Mb of memory will be written (except the busy part of the stack). Therefore, do not call this function too frequently, otherwise your performance will suffer.

Use with caution when stack size is greatly increased.

Examples

Code (Delphi)

```
procedure TForm1.ClearStackOnIdle(Sender: TObject; var Done: Boolean);
begin
   if CanCallMemWipeStack then
      MemWipeStack;
   Done := True;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
   Application.OnIdle := Self.ClearStackOnIdle;
end;
```

See also

- CanCallMemWipeStack 328
- EnsureThreadStack 387

10.219 MonoFont

Returns fixed-width font.

Unit

EFonts 95

Syntax

Code (Delphi)

```
function MonoFont: TFont;
```

Return value

Fixed-pitch (monospace) system font.

Remarks

This function returns font from GetStockObject(ANSI_FIXED_FONT).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- GUIFont 432

10.220 NameThread

Names thread for debugging.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure NameThread(
   const AName: String;
   const AThreadID: Cardinal = 0
);
```

Parameters

AName [in]

A thread name and/or description to be set for the thread identified via `AThreadID` argument. Can be arbitrary string. Thread name does not have to be unique.

AThreadID [in, optional]

Thread ID (TID) of the thread which name has to be set.

Remarks

The name (description) of a thread can be set more than once; the most recently set value is used. You can retrieve the description of a thread via TEurekaThreadData.Name 225.

Note

It is recommended to use EurekaLog's naming routines instead of RTL (and possibly other frameworks), because EurekaLog supports Get/SetThreadDescription API, while RTL does not.

10.221 NameThreadForDebugging

Alias for `NameThread`, this overrides System.Classes.TThread.NameThreadForDebugging.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure NameThreadForDebugging(
   const AName: String;
   const AThreadID: Cardinal = 0
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AName [in]
Thread's name.

AThreadID [in, optional]
TID (Thread ID) of the thread to set new name.

Remarks

See NameThread 496 for a detailed description. `NameThreadForDebugging` is the same function as `NameThread`.

See also

- NameThread Function 496

10.222 PageSize

Returns current memory's page size.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function PageSize: Cardinal;
inline;
```

Return value

Page size in bytes. Default is 4Kb (4096).

Remarks

The page size and the granularity of page protection and commitment. This is the page size used by the VirtualAlloc function.

This is a `dwPageSize` field from the GetSystemInfo function.

See also

- AllocationGranularity 322
- AllocatePageAlign 321

10.223 PauseFreezeCheck

Temporarily disables freeze checks.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
procedure PauseFreezeCheck;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function will temporarily stop freeze detection without shutdowning background freeze detection thread.

Use this function to temporarily turn off freeze detection during executing long lasting blocking code.

Warning

Each call to `PauseFreezeCheck` must be paired with a call to ResumeFreezeCheck 524.

Tip

`PauseFreezeCheck` can be called multiple times by the same thread.

Note

`PauseFreezeCheck` can be called from any thread.

See also

- FreezeNotifyStillAlive Function 401
- ResumeFreezeCheck Function 524
- IsFreezeCheckInstalled Function 458
- DoneCheckFreeze Function 357
- TFreezeThread.ChecksPaused Property 247

10.224 PrimaryInstance

The same as MainInstance. See remarks for more info.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
function PrimaryInstance: HINST;
```

Return value

The instance handle for the main executable.

Remarks

This is a replacement for MainInstance. In some rare cases MainInstance may be 0 instead of actual value. `PrimaryInstance` can be used as workaround for such cases.

Do not use MainInstance when working with EurekaLog. Use `PrimaryInstance` instead.

See also

- MainInstance
- CurrentInstance 348
- HInstance

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.225 RaiseExpected

Raises expected exception.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure RaiseExpected(
   const AException: TObject;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID = 0
); overload;

procedure RaiseExpected(
   const AException: TObject;
   const AURL: String;
   const ABugID: TBugID = 0
); overload;
```

Parameters

AException [in]
Exception object to raise.

AContextNumber [in, optional]
Help context number for this exception.

ABugID [in, optional]
BugID for this exception.

AURL [in, optional]
URL for this exception.

Remarks

Raises exception and marks it as "expected". Additionally, it sets ExpectedNumber/ ExpectedURL 197 and BugID 187 properties.

See also

- RaiseOuterExceptionExpected Function 501
- MarkExpected Function 480
- ShowExpected Function 557
- IsExpected Function 455

10.226 RaiseFreezeException

Raises freeze/hang/deadlock exception for the specified thread.

Unit

EFreeze 96

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure RaiseFreezeException(
   const AThreadID: Cardinal = 0;
   const AThreadHandle: THandle = 0;
   const AAntiFreezeMsg: String = ''
);
```

Parameters

AThreadID [in, optional]
Thread ID to raise exception for. Default is 0 - which means "use MainThreadID".

AThreadHandle [in, optional]
A handle of thread identified by `AThreadID`.

AAntiFreezeMsg [in, optional]
A message for the exception. Default is `'The application seems to be frozen.'`. It is recommended to pass `CurrentEurekaLogOptions.CustomizedExpandedTexts[mtException_AntiFreeze]` here.

Remarks

Function will construct `EFrozenApplication` exception, build call stack for the thread identified by `AThreadID` (and `AThreadHandle`), and then invokes standard EurekaLog processing (i.e. create bug report, show error dialog, send report, etc.).

Note

This function is used by freeze detection thread to raise exception when freeze is detected.

Note

RaiseFreezeException can be called from any thread.

See also

- InitCheckFreeze Function 436
- TFreezeThread Class 246
- FreezeThreadClass Variable 756

10.227 RaiseOuterException

Raises an exception and captures the current one.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure RaiseOuterException(
   const E: TObject
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
);
```

Parameters

E [in]
Exception object to raise.

Remarks

Call `RaiseOuterException` from within an except block to raise a new exception and preserve the current one. `RaiseOuterException` raises a new exception and sets the current one as an inner exception. `RaiseOuterException` is useful when creating exception chains.

Note

The `RaiseOuterException` function is the same as Exception.RaiseOuterException or Exception.ThrowOuterException, except it works on any exception object.

Note

Consider using the function 501 instead.

See also

- RaiseOuterExceptionExpected Function 501

10.228 RaiseOuterExceptionExpected

Raises expected exception and captures the current one.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure RaiseOuterExceptionExpected(
   const AException: TObject;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID = 0
); overload;

procedure RaiseOuterExceptionExpected(
   const AException: TObject;
   const AURL: String;
   const ABugID: TBugID = 0
); overload;
```

Parameters

AException [in]
Exception object to raise.

AContextNumber [in, optional]
Help context number for this exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ABugID [in, optional]
BugID for this exception.

AURL [in, optional]
URL for this exception.

Remarks

Call `RaiseOuterException` from within an except block to raise a new exception and preserve the current one. `RaiseOuterException` raises a new exception and sets the current one as an inner exception. `RaiseOuterException` is useful when creating exception chains.

`RaiseOuterException` raises exception and marks it as "expected". Additionally, it sets ExpectedNumber/ ExpectedURL 197 and BugID 187 properties.

See also

- RaiseExpected Function 499
- MarkExpected Function 480
- ShowExpected Function 557
- IsExpected Function 455

10.229 RegisterDialogClass

Registers dialog class.

Unit

EDialog 72

Syntax

Code (Delphi)

```
procedure RegisterDialogClass(
   const AClass: TBaseDialogClass 611
);
```

Parameters

AClass [in]
Dialog class to register.

Remarks

Function registers the specified dialog class into global list of available dialog classes. Each dialog class must be registered before using. Once registered, dialog class can be used (selected) via ExceptionDialogType property of TEurekaModuleOptions 216.

Note

This function adds dialog class to the end of the list.

Note

Once registered, dialog class can not be unregistered.

Note

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

There is no limitation to register different dialog classes with the same class name. Double-registering the same class will have no effect.

See also

- RegisterDialogClassFirst 503
- TBaseDialog 36
- FindDialogClass 399
- CurrentEurekaLogOptions

10.230 RegisterDialogClassFirst

Registers dialog class.

Unit

EDialog 72

Syntax

Code (Delphi)

```
procedure RegisterDialogClassFirst(
   const AClass: TBaseDialogClass 611
);
```

Parameters

AClass [in]

Dialog class to register.

Remarks

Function registers the specified dialog class into global list of available dialog classes. Each dialog class must be registered before using. Once registered, dialog class can be used (selected) via ExceptionDialogType property of TEurekaModuleOptions 216.

Note

This function adds dialog class to the beginning of the list.

Note

Once registered, dialog class can not be unregistered.

Note

There is no limitation to register different dialog classes with the same class name. Double-registering the same class will have no effect.

See also

- RegisterDialogClass 502
- TBaseDialog 36
- FindDialogClass 399
- CurrentEurekaLogOptions

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.231 RegisterEventAttachedFilesRequest

Registers OnAttachedFilesRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventAttachedFilesRequest(
   const AHandler: TELEvAttachedFilesRequestMeth 21;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventAttachedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvAttachedFilesRequestProc 625;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 587

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.232 RegisterEventBeginReportGen

Registers OnBeginReportGen event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventBeginReportGen(
   const AHandler: TELEvReportGenMeth 30;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventBeginReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 657;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 588

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.233 RegisterEventCustomBugID

Registers OnCustomBugID event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventCustomBugID(
   const AHandler: TELEvCustomBugIDMeth 25;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomBugID(
   const ACustom: Pointer;
   const AHandler: TELEvCustomBugIDProc 626;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 589

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.234 RegisterEventCustomButtonClick

Registers OnCustomButtonClick event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventCustomButtonClick(
   const AHandler: TELEvCustomButtonClickMeth 28;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvCustomButtonClickProc 628;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 589

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.235 RegisterEventCustomDataRequest

Registers OnCustomDataRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventCustomDataRequest(
   const AHandler: TELEvCustomDataRequestMeth 26;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomDataRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomDataRequestProc 631;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 590
- AddCustomDataToBugReport function 319

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.236 RegisterEventCustomFileName

Registers OnCustomFileName event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventCustomFileName(
   const AHandler: TELEvCustomFileNameMeth 24;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomFileName(
   const ACustom: Pointer;
   const AHandler: TELEvCustomFileNameProc 633;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 591

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.237 RegisterEventCustomWebFieldsRequest

Registers OnCustomWebFieldRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventCustomWebFieldsRequest(
   const AHandler: TELEvCustomWebFieldsRequestMeth 27;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomWebFieldsRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomWebFieldsRequestProc 635;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 592
- AddCustomWebField function 320

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.238 RegisterEventEndReportGen

Registers OnEndReportGen event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventEndReportGen(
   const AHandler: TELEvReportGenMeth 30;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventEndReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 657;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 593

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.239 RegisterEventExceptionAction

Registers OnExceptionAction event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionAction(
   const AHandler: TELEvExceptionActionMeth 16;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionAction(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionActionProc 638;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 594

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.240 RegisterEventExceptionError

Registers OnExceptionError event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionError(
   const AHandler: TELEvExceptionErrorMeth 18;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionError(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionErrorProc 642;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 594

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.241 RegisterEventExceptionNotify

Registers OnExceptionNotify event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionNotify(
   const AHandler: TELEvExceptionNotifyMeth 14;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionNotify(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionNotifyProc 645;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 595

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.242 RegisterEventHelpButtonClick

Registers OnHelpButtonClick event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventHelpButtonClick(
   const AHandler: TELEvHelpButtonClickMeth 29;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventHelpButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvHelpButtonClickProc 648;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 596

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.243 RegisterEventPasswordRequest

Registers OnPasswordRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventPasswordRequest(
   const AHandler: TELEvPasswordRequestMeth 19;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventPasswordRequest(
   const ACustom: Pointer;
   const AHandler: TELEvPasswordRequestProc 651;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 597

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.244 RegisterEventRaise

Registers OnRaise event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventRaise(
   const AHandler: TELEvRaiseMeth 12;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventRaise(
   const ACustom: Pointer;
   const AHandler: TELEvRaiseProc 653;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 598

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.245 RegisterEventReproduceRequest

Registers OnReproduceRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventReproduceRequest(
   const AHandler: TELEvReproduceRequestMeth 23;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventReproduceRequest(
   const ACustom: Pointer;
   const AHandler: TELEvReproduceRequestProc 659;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 598

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.246 RegisterEventZippedFilesRequest

Registers OnZippedFilesRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure RegisterEventZippedFilesRequest(
   const AHandler: TELEvZippedFilesRequestMeth 22;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventZippedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvZippedFilesRequestProc 661;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

AFirst [in, optional]
`True` - insert handler as first, `False` - insert handler as last (default).

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

This pointer is not used by EurekaLog, so you can use it for your own goals.

Can be `nil` if not used.

Remarks

Please see description of event handler type for more details about event itself.

Please see unit description for common event concept details.

Tip

It's best to call this routine in object's constructor (for method event handlers) or unit's initialization section (for procedural event handlers).

See also

- General events concept 83
- Unregistering event handler 599

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.247 RegisterSender

Registers send method class.

Unit

ESend 120

Syntax

Code (Delphi)

```
procedure RegisterSender(
   const ASenderClass: TELUniversalSenderClass 669
);
```

Parameters

AClass [in]

Send method class to register.

Remarks

Function registers the specified send method class into global list of available send method classes. Each send method class must be registered before using. Once registered, send method class can be used (selected) via SenderClasses property of TEurekaModuleOptions 216 (which is usually altered using AddSenderClass and RemoveSenderClass methods).

Note

This function adds send method class to the end of the list.

Note

Once registered, send method can not be unregistered.

Note

There is no limitation to register different send method classes with the same class name. Double-registering the same class will have no effect.

See also

- RegisterSenderFirst 520
- TELUniversalSender 171
- FindSender 401
- EnumSenders 387
- CurrentEurekaLogOptions

10.248 RegisterSenderFirst

Registers send method class.

Unit

ESend 120

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure RegisterSenderFirst(
   const ASenderClass: TELUniversalSenderClass 669
);
```

Parameters

AClass [in]

Send method class to register.

Remarks

Function registers the specified send method class into global list of available send method classes. Each send method class must be registered before using. Once registered, send method class can be used (selected) via SenderClasses property of TEurekaModuleOptions 216 (which is usually altered using AddSenderClass and RemoveSenderClass methods).

Note

This function adds send method class to the beginning of the list.

Note

Once registered, send method class can not be unregistered.

Note

There is no limitation to register different send method classes with the same class name. Double-registering the same class will have no effect.

See also

- RegisterSender 520
- TELUniversalSender 171
- FindSender 401
- EnumSenders 387
- CurrentEurekaLogOptions

10.249 RegKeyCheck

Checks registry key for existing.

Unit

EConfig 63

Syntax

```
function RegKeyCheck(
   const ARoot: HKey;
   const AKey: String
): Boolean;
```

Parameters

ARoot

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

One of predefined HKEY_XYZ constants. Usually it's HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE.

AKey
A full absolute path to key.

Return value
True, if specified key exists (i.e. you have a minimum a read access), False - otherwise.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

See also
- RegKeyRead 522
- RegKeyWrite 523

10.250 RegKeyRead

Reads string from registry key.

Unit
EConfig 63

Syntax

Code (Delphi)

```
function RegKeyRead(
   const ARoot: HKey;
   const AKey, AName: String;
   const ADefault: String = ''
): String;
```

Parameters
ARoot
One of predefined HKEY_XYZ constants. Usually it's HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE.

AKey
A full absolute path to key.

AName
A name of value to read.

ADefault
Optional default value.

Return value
Actual value from registry key, if it's exists. `ADefault` value - if there is error (no access, value or key do not exist).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Returned value does not need to be decoded by DecodeConfigValue 352, because registry can hold exact binary copy of data itself.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

See also

- RegKeyCheck 521
- RegKeyWrite 523
- ConfigReadString 338

10.251 RegKeyWrite

Writes string to registry key.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure RegKeyWrite(
   const ARoot: HKey;
   const AKey, AName, AValue: String
); overload;
```

Parameters

ARoot

One of predefined HKEY_XYZ constants. Usually it's HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE.

AKey

A full absolute path to key.

AName

A name of value to write.

AValue

A value to write. Mandatory, but can be an empty string. Writing empty string will not delete value.

This value does not need to be encoded by EncodeConfigValue 384, because registry can hold exact binary copy of data itself.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Warning
This is low-level routine. Use it only to implement your own custom high-level wrapper. Use ConfigReadString 338 or ConfigWriteString 342 (or their equivalents) to read and write configurations.

See also

- RegKeyCheck 521
- RegKeyRead 522
- ConfigWriteString 342

10.252 ResumeFreezeCheck

Enables freeze checks back.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
procedure ResumeFreezeCheck;
```

Remarks

This function will enable freeze detection previously disabled by a call to PauseFreezeCheck 497.

Note
Freeze detection is not necessary enabled after a call to this function. For example, if `PauseFreezeCheck` was called twice - then freeze detection checks will be enabled only after second call to `ResumeFreezeCheck`.

Note
`ResumeFreezeCheck` can be called from any thread.

See also

- PauseFreezeCheck Function 497
- IsFreezeCheckEnabled Function 457
- InitCheckFreeze Function 436

10.253 RSADecrypt

Decrypts encrypted data by using RSA cipher.

Unit

EEncrypt 80

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure RSADecrypt(
   const AKey: TRSAKey[719];
   const AEncryptedData: Pointer;
   out AData: Pointer;
   var ADataSize: Cardinal
); overload;

function RSADecrypt(
   const AKey: TRSAKey[719];
   const AEncryptedData: RawByteString
): RawByteString; overload;

procedure RSADecrypt(
   const AKey: TRSAKey[719];
   var AInPlaceData: TEncryptBuffer[671]
); overload;
```

Parameters

AKey [in]
A key for decryption. Comes from RSAGenKey[531] function or RSALoadPrivateKey[532] function. A private key will be used.

AEncryptedData [in]
A data to decrypt. Must be in little-endian format.

AData [out]
A decrypted AEncryptedData's "clear text" data. Must be deleted with FreeMem function (but we recommend to use SecureFree[545] function instead). Will have different size from AData.

ADataSize [in]
On input: size of AEncryptedData.
On output: size of AData.

AEncryptedData [in]
Encrypted data (in string form) to decrypt. Must be in little-endian format.

AInPlaceData [in, out]
On input: encrypted buffer to decrypt. Must be in little-endian format.
On output: "clear text" buffer (same size).

Return value

Decrypted "clear text" data.

Remarks

This function decrypts data previosly encrypted with RSAEncrypt[528] function with private key of asymmetric RSA cipher. The encrypted data is passed in AEncryptedData parameter. The function outputs decrypted data back to AData parameter. AData expected to have different size from AEncryptedData. In other words, asymmetric decryption with RSA does change the size of the data.

The overload variant with RawByteString decrypts AEncryptedData parameter and outputs decrypted data as function's result.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

The overload variant with `AInPlaceData` decrypts data "in place".

Important!

Asymmetric chipers are slow. Therefore, they are not designed to encrypt/decrypt large amount of data. The usual approach is to use asymmetric chiper to encrypt/decrypt another key - a key for some symmetric cipher, and then use (fast) symmetric cipher to encrypt/decrypt large amounts of data.

Tip

You always use public key for encryption and private key for decryption with asymmetric cipher - and not the other way around. You should use digital signing and verification for reversed order. I.e. you sign data by using your private key and verify signature by using your public key.

Examples

Code (Delphi)

```
// This example decrypts file encrypted in code example for RSAEncrypt
// The source file contains encrypted (random) symmetric key in
// the beginning of the file, followed by encrypted data.
var
   // (Random) symmetric key to decrypt file
   TEAKey: TTEAKey;
   // Asymmetric key to encrypt symmetric key
   RSAKey: TRSAKey;
   // Encrypted symmetric key
   EncryptedKey: TEncryptBuffer;
   // File/data to decrypt
   Data: TMemoryStream;
   // Wrapper for "in place" decryption
   DataToDecrypt: TEncryptBuffer;
   // Stream to read encrypted data
   FS: TFileStream;
begin
   FS := TFileStream.Create('C:\Test\Text.bin', fmOpenRead or fmShareDer
   try

      try
         // First, read encrypted (random) symmetric key from source
         FS.ReadBuffer(EncryptedKey.cbData, SizeOf(EncryptedKey.cbData))
         EncryptedKey.pbData := AllocMem(EncryptedKey.cbData);
         try
          FS.ReadBuffer(EncryptedKey.pbData^, EncryptedKey.cbData);

          // Decrypt symmetric key with asymmetric (private) key
          RSAKey := RSALoadPrivateKey('C:\Private.pem', rsText, 'Private
          try
          EEncrypt.RSADecrypt(RSAKey, EncryptedKey);
          finally
          RSADestroyKey(RSAKey);
          end;

          // Copy unencrypted buffer into actual key
          Assert(EncryptedKey.cbData = SizeOf(TEAKey));
          Move(EncryptedKey.pbData^, TEAKey, SizeOf(TEAKey));
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
         finally
          SecureFree(EncryptedKey);
         end;

         // Now we have symmetric key to decrypt actual file's content (

         Data := TMemoryStream.Create;
         try
          // The rest of the file is the actual data to decrypt
          Data.CopyFrom(FS, FS.Size - FS.Position);

          // Decrypt Data in place
          DataToDecrypt.pbData := Data.Memory;
          DataToDecrypt.cbData := Data.Size;

          // This does not alter Data.Memory and Data.Size
          // It simply changes content of Data.Memory^
          TEADecrypt(TEAKey, DataToDecrypt);

          // Symmetric key is no longer needed
          TEADestroyKey(TEAKey); // - same as SecureFree(TEAKey);

          // File/data is decrypted - save final result into file
          Data.SaveToFile('C:\Text.txt');
         finally
          SecureFree(Data);
         end;

      finally
         TEADestroyKey(TEAKey); // - same as SecureFree(TEAKey);
      end;
   finally
      FreeAndNil(FS);
   end;
end;
```

See also

- TEADeriveKey Function 565
- TEAEncrypt Function 572

10.254RSADestroyKey

Deletes RSA key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure RSADestroyKey(
   var AKey: TRSAKey 719
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure RSADestroyKey(
   var AKeyPart: TEncryptBuffer 671
); overload;
```

Parameters

AKey [in]
RSA key to be released.

AKeyPart [in]
Public or private key to be released.

Remarks

This function deletes RSA key allocated by RSAGenKey 531 or RSALoadPublicKey 533/RSALoadPrivateKey 532 functions.

Call this function to release RSA key once you have finished working with it.

See also

- RSAGenKey Function 531
- RSALoadPublicKey Function 533
- RSALoadPrivateKey Function 532

10.255RSAEncrypt

Encrypts "clear text" data by using RSA cipher.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure RSAEncrypt(
   const AKey: TRSAKey 719;
   const AData: Pointer;
   out AEncryptedData: Pointer;
   var ADataSize: Cardinal
); overload;

function RSAEncrypt(
   const AKey: TRSAKey 719;
   const AData: RawByteString
): RawByteString; overload;

procedure RSAEncrypt(
   const AKey: TRSAKey 719;
   var AInPlaceData: TEncryptBuffer 671
); overload;
```

Parameters

AKey [in]
A key for encryption. Comes from RSAGenKey 531 function or RSALoadPublicKey 533 function. A public

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

key will be used.

AData [in]
A data to encrypt.

AEncryptedData [out]
An encrypted AData's data. Must be deleted with FreeMem function (but we recommend to use SecureFree 545 function instead). Will have different size from AData. Data is returned in little-endian format.

ADataSize [in, out]
On input: size of AData.
On output: size of AEncryptedData (will be different).

AData [in]
String "clear text" data to encrypt.

AInPlaceData [in, out]
On input: "clear text" buffer to encrypt.
On output: encrypted buffer (same size, little-endian).

Return value

Encrypted AData's data in a string form. Data is returned in little-endian format.

Remarks

This function encrypts data passed in AData parameter with public key of asymmetric RSA cipher and outputs encrypted data back to AEncryptedData parameter. AEncryptedData expected to have different size from original AData. In other words, asymmetric encryption with RSA does change the size of the data.

The overload variant with RawByteString encrypts AData parameter and outputs encrypted data as function's result.

The overload variant with AInPlaceData encrypts data "in place".

Important!

Asymmetric chipers are slow. Therefore, they are not designed to encrypt/decrypt large amount of data. The usual approach is to use asymmetric chiper to encrypt/decrypt another key - a key for some symmetric cipher, and then use (fast) symmetric cipher to encrypt/decrypt large amounts of data.

Tip

You always use public key for encryption and private key for decryption with asymmetric cipher - and not the other way around. You should use digital signing and verification for reversed order. I.e. you sign data by using your private key and verify signature by using your public key.

Examples

Code (Delphi)

```
var
   // (Random) symmetric key to encrypt file
   TEAKey: TTEAKey;
   // Asymmetric key to encrypt symmetric key
   RSAKey: TRSAKey;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   // Encrypted symmetric key
   EncryptedKey: Pointer; DataSize: Cardinal;
   // File/data to encrypt
   Data: TMemoryStream;
   // Wrapper for "in place" encryption
   DataToEncrypt: TEncryptBuffer;
   // Stream to save final result
   FS: TFileStream;
begin
   // Create strong random session key
   // It is best to use cryptographic-quality random generator
   // This key will be used to encrypt file
   TEAKey := TEAInitSessionKey;
   try

      // First, encrypt symmetric key with asymmetric key

      // FS is used twice:
      // 1). To write encrypted symmetric key
      // 2). To write encrypted file/data itself
      FS := nil;
      try

         // Load public key from file
         RSAKey := RSALoadPublicKey('C:\Test\Public.pem', rsText);
         try
          // Encrypt the TEAKey key into EncryptedKey buffer
          DataSize := SizeOf(TEAKey);
          EEncrypt.RSAEncrypt(RSAKey, @TEAKey, EncryptedKey, DataSize);
          try
          // Once symmetric key is encrypted - we no longer need asymmet
          RSADestroyKey(RSAKey);

          // Now EncryptedKey of DataSize bytes holds the encrypted symm

          // Simply store encrypted key in the encrypted file itself
          FS := TFileStream.Create('C:\Text.bin', fmCreate or fmShareExc
          FS.WriteBuffer(DataSize, SizeOf(DataSize));
          FS.WriteBuffer(EncryptedKey^, DataSize);

          // We did not finished writing into FS yet...

          finally
          SecureFree(EncryptedKey, DataSize);
          end;
         finally
          RSADestroyKey(RSAKey);
         end;

         // Next, encrypt the file itself

         Data := TMemoryStream.Create;
         try

          // Load C:\Text.txt file into Data
          Data.LoadFromFile('C:\Text.txt');
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
          // Encrypt Data in place
          DataToEncrypt.pbData := Data.Memory;
          DataToEncrypt.cbData := Data.Size;

          // This does not alter Data.Memory and Data.Size
          // It simply changes content of Data.Memory^
          TEAEncrypt(TEAKey, DataToEncrypt);

          // Data is encrypted, so save it to C:\Text.bin
          FS.CopyFrom(Data, Data.Size);

        finally
          // Wipe memory and release stream object
          SecureFree(Data);
        end;

      finally
        FreeAndNil(FS);
      end;

      // Now C:\Text.bin is ready
      // It contains encrypted (random) symmetric key and encrypted data
      // It can only be decrypted by the one having private RSA key
      // See code example for RSADecrypt

   finally
      TEADestroyKey(TEAKey); // - same as SecureFree(TEAKey);
   end;
end;
```

See also

- RSADecrypt Function 524
- RSAGenKey Function 531
- RSALoadPublicKey Function 533

10.256 RSAGenKey

Creates new RSA key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function RSAGenKey(
   const AKeyBitSize: Cardinal = 4096
): TRSAKey 719;
```

Parameters

AKeyBitSize [in, opt]

Size of new key in bits. Default is 4096. Minimum is 384. Maximum is 16384. Must be a factor of 8.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Generated RSA key.

Delete this key by using RSADestroyKey 527 function.

Remarks

This function generates a new RSA key (public/private keys pair). The returned key must be deleted by RSADestroyKey 527 function once you have finished working with it.

Resulting key may be used for encryption using RSAEncrypt 528 function or for digital signature using RSASign 536 function.

Resulting key may be exported by using RSASavePublicKey 535 and RSASavePrivateKey 534 functions.

You may import previosly created key instead of generating new key. Use RSALoadPublicKey 533 and/or RSALoadPrivateKey 532 functions.

RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. RSA is a relatively slow algorithm, and because of this it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

Important!
It is currently recommended that key size be at least 2048 bits long for security reasons.

See also

- RSALoadPublicKey Function. 533
- RSALoadPrivateKey Function. 532
- RSADestroyKey Function. 527
- RSAEncrypt Function. 528
- RSASign Function. 536
- RSASavePublicKey Function. 535
- RSASavePrivateKey Function. 534

10.257 RSALoadPrivateKey

Imports private key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function RSALoadPrivateKey(
   const AFileName: String;
   const AType: TRSAImport 718;
   const APassword: String = ''
): TRSAKey 719; overload;

function RSALoadPrivateKey(
   const AStream: Classes.TStream;
   const AType: TRSAImport 718;
   const APassword: String = ''
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): TRSAKey[719]; overload;
```

Parameters

AFileName [in]
A fully qualified absolute file name. File must exist. The file is usually obtained from RSASavePrivateKey[534] function.

AType [in]
AFileName/AStream's type. It must match type specified in RSASavePrivateKey[534] when exporting the key.

AStream [in]
A stream to load private key.

Return value

RSA private key. Public key part will be unassigned. This key must be released with RSADestroyKey[527] function.

Remarks

This function loads private key from given file or stream.

See also

- RSASavePrivateKey Function[534]
- RSALoadPublicKey Function[533]
- RSADestroyKey Function[527]
- RSADecrypt Function[524]
- RSASign Function[536]

10.258 RSALoadPublicKey

Imports public key.

Unit

EEncrypt[80]

Syntax

Code (Delphi)

```
function RSALoadPublicKey(
   const AFileName: String;
   const AType: TRSAImport[718]
): TRSAKey[719]; overload;

function RSALoadPublicKey(
   const AStream: Classes.TStream;
   const AType: TRSAImport[718]
): TRSAKey[719]; overload;
```

Parameters

AFileName [in]
A fully qualified absolute file name. File must exist. The file is usually obtained from RSASavePublicKey[535] function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AType [in]
AFileName/AStream's type. It must match type specified in RSASavePublicKey 535 when exporting the key.

AStream [in]
A stream to load public key.

Return value
RSA public key. Private key part will be unassigned. This key must be released with RSADestroyKey 527 function.

Remarks
This function loads public key from given file or stream.

See also
- RSASavePublicKey Function 535
- RSALoadPrivateKey Function 532
- RSADestroyKey Function 527
- RSAEncrypt Function 528
- RSAVerify Function 538

10.259 RSASavePrivateKey

Exports private key.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
procedure RSASavePrivateKey(
   const AKey: TRSAKey 719;
   const AFileName: String;
   const AType: TRSAExport 717;
   const APassword: String = ''
); overload;

procedure RSASavePrivateKey(
   const AKey: TRSAKey 719;
   const AStream: Classes.TStream;
   const AType: TRSAExport 717;
   const APassword: String = ''
); overload;
```

Parameters
AKey [in]
RSA key to export. Must come from RSAGenKey 531 or RSALoadPrivateKey 532 functions.

AFileName [in]
A fully qualified absolute file name. New file will be created. Recommended file's extensions are '.blob', '.der', or 'pem'.

AType [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AFileName/AStream's type.

APassword [in, opt]
Optional password to encrypt private key before exporting.

AStream [in]
A stream to save private key.

Remarks

This function saves private key from given RSA key to file or stream.

Key can be imported later with RSALoadPrivateKey 532 function.

See also

- RSALoadPublicKey Function 533
- RSASavePublicKey Function 535
- RSAGenKey Function 531
- RSADecrypt Function 524
- RSASign Function 536

10.260 RSASavePublicKey

Exports public key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure RSASavePublicKey(
   const AKey: TRSAKey 719;
   const AFileName: String;
   const AType: TRSAExport 717
); overload;

procedure RSASavePublicKey(
   const AKey: TRSAKey 719;
   const AStream: Classes.TStream;
   const AType: TRSAExport 717
); overload;
```

Parameters

AKey [in]
RSA key to export. Must come from RSAGenKey 531 or RSALoadPublicKey 533 functions.

AFileName [in]
A fully qualified absolute file name. New file will be created. Recommended file's extensions are '.blob', '.der', or 'pem'.

AType [in]
AFileName/AStream's type.

AStream [in]
A stream to save public key.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function saves public key from given RSA key to file or stream.

Key can be imported later with RSALoadPublicKey 533 function.

See also

- RSALoadPublicKey Function 533
- RSASavePrivateKey Function 534
- RSAGenKey Function 531
- RSAEncrypt Function 528
- RSAVerify Function 538

10.261 RSASign

Signs data.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function RSASign(
   const AKey: TRSAKey 719;
   const AData: Pointer;
   const ADataSize: Cardinal
): TRSASignature 720; overload;

function RSASign(
   const AKey: TRSAKey 719;
   const AData: TEncryptBuffer 671
): TRSASignature 720; overload;

function RSASign(
   const AKey: TRSAKey 719;
   const AData: RawByteString
): TRSASignature 720; overload;
```

Parameters

AKey [in]

A key for digital signing. Comes from RSAGenKey 531 function or RSALoadPrivateKey 532 function. A private key will be used.

AData [in]

Data to be signed.

ADataSize [in]

Size of AData in bytes.

Return value

Digital signature (little-endian). Release it with SecureFree 545 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function digitally signs the specified data and returns digital signature. Digital signature is a small RAW bytes value, which can be freely copied and/or exported/imported.

Your private key will be used for digital signing.

You can verify the returned signature with RSAVerify 538 function.

Note
Internally `RSASign` will calculate SHA-1 hash of `AData` and sign (encrypt) resulting hash with RSA and private key. EMSA-PKCS1-v1_5 scheme is used for padding.

Important!
The resulting signature bytes are in little-endian order. You must reverse bytes order if you want to verify this signature with other tools which requires big-endian order (such as .NET or OpenSSL).

Examples

Code (Delphi)

```
var
   // Key to sign data
   RSAKey: TRSAKey;
   // Data to sign
   S: String;
   // Digital signature for S
   Signature: TRSASignature;
begin
   // Load private key
   RSAKey := RSALoadPrivateKey('C:\Private.pem', rsText, 'Private key pa
   try

      // Data to sign
      S := 'Some data';

      // Sign data
      Signature := RSASign(RSAKey, Pointer(S), Length(S) * SizeOf(Char))
      try
         // Clear key as soon as possible
         RSADestroyKey(RSAKey);

         // Store signature somewhere
         edSignature.Text := HexEncodeToString(Signature.pbData, Signatu

      finally
         // Free signature once we finished working with it
         SecureFree(Signature);
      end;
   finally
      RSADestroyKey(RSAKey);
   end;
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- RSAVerify Function 538
- RSAGenKey Function 531
- RSALoadPrivateKey Function 532

10.262RSAVerify

Verifies digital signature.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function RSAVerify(
   const AKey: TRSAKey 719;
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASignature: Pointer;
   const ASignatureSize: Cardinal
): Boolean; overload;

function RSAVerify(
   const AKey: TRSAKey 719;
   const AData: TEncryptBuffer 671;
   const ASignature: TRSASignature 720
): Boolean; overload;

function RSAVerify(
   const AKey: TRSAKey 719;
   const AData: RawByteString;
   const ASignature: TRSASignature 720
): Boolean; overload;
```

Parameters

AKey [in]
A key for verification. Comes from RSAGenKey 531 function or RSALoadPublicKey 533 function. **Must** match the key used in RSASign 536 function to sign `AData`. A public key will be used.

AData [in]
Data to be verified (the same data as the one passed to RSASign 536 function). Must be unmodified (otherwise function will return `False`).

ADataSize [in]
Size of `AData` in bytes.

ASignature [in]
Signature to be verified (the same signature as the one returned by RSASign 536 function). Little-endian.

ASignatureSize [in]
Size of `ASignature` in bytes.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

`True` - if digital signature is OK, `False` - if digital signature is broken (i.e. `AData` is not the same/was changed; or wrong key is used).

Remarks

This function verifies digital signature created by RSASign 536 function.

Your public key will be used for verification.

Important!

The signature bytes must be in little-endian order. You must reverse bytes order (before calling `RSAVerify`) if you want to verify signature generated by other tools which produce big-endian order signatures (such as .NET or OpenSSL).

Examples

Code (Delphi)

```
var
   // Key to verify
   RSAKey: TRSAKey;
   // Data to verify
   S: String;
   // Digital signature for S
   Signature: TRSASignature;
   // If signature is broken or not
   SignatureOK: Boolean;
begin
   // Load public key
   RSAKey := RSALoadPublicKey('C:\Public.pem', rsText);
   try

      // Data to verify
      // (same as in RSASign example)
      S := 'Some data';

      // Load signature
      Signature.cbData := HexCalcDecodedSize(Length(edSignature.Text));
      Signature.pbData := AllocMem(Signature.cbData);
      try
         Base64DecodeFromString(edSignature.Text, Signature.pbData);

         // Verify signature
         SignatureOK := RSAVerify(RSAKey, Pointer(S), Length(S) * SizeOf

         // Clear key as soon as possible
         RSADestroyKey(RSAKey);

         if SignatureOK then
          // OK for S = 'Some data'
          Application.MessageBox('OK', 'Signature verification', MB_OK)
         else
          // FAIL for any other S
          Application.MessageBox('FAIL', 'Signature verification', MB_OK
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
      finally
         SecureFree(Signature);
      end;
   finally
      RSADestroyKey(RSAKey);
   end;
end;
```

See also

- RSASign Function 536
- RSAGenKey Function 531
- RSALoadPublicKey Function 533

10.263 SafeExec

Executes routine with exception handling. Any exception is handled as fatal exception.

Unit

ELowLevel 105

Syntax

Code (Delphi)

```
procedure SafeExec(
   const AProc: ETypes.TProcedure;
   const ASection: String
);
```

Parameters

AProc [in]

A procedure to execute.

Must not throw exceptions as normal operation. Any exception inside this procedure is considered as fatal.

ASection [in]

Fully qualified name of procedure. **Must** have form `'UnitName.ProcName'`.

ProcName **must** be either `'Init'` or `'Done'`.

Remarks

This routine is used to wrap code from `initialization` / `finalization` sections for safe executing.

Passed procedure is either `Init` or `Done` procedures, which performs code which should be in `initialization` and `finalization` sections. See examples for exact pattern of usage.

Any exception inside specified procedure is considered as fatal exception and triggers an internal EurekaLog's error.

Note

The names of routines (`'Init'` and `'Done'`) are important, because this procedure also changes

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

IntoInitialization and IntoFinalization variables.

Any exception inside this procedure is handled by _InitDoneErrorHandler (if assigned).

Examples

Code (Delphi)

```
unit ESomeUnit;
// ...
implementation
// ...
procedure Init;
begin
   // ...
end;
procedure Done;
begin
   // ...
end;
initialization
   SafeExec(Init, 'ESomeUnit.Init');
finalization
   SafeExec(Done, 'ESomeUnit.Done');
end.
```

Warning

Any EurekaLog's unit **must** follow this pattern above (except those which do not have init/done code).

Important!

You must follow this pattern **exactly**.

No other code allowed in `initialization` / `finalization` section.

No other names allowed for `Init` / `Done` procedures.

See also

- _InitDoneErrorHandler
- ShowInternalErrorMessage

10.264 Screenshot

Takes screenshot.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
procedure Screenshot(
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const ARect: TRect;
   const AFocusWnd: HWND;
   const ACursorPos: TPoint;
   const ACursor: HCURSOR;
   const AFileName: String;
   const A8Bit: Boolean = True
); overload;

procedure Screenshot(
   const ARect: TRect;
   const AFocusWnd: HWND;
   const ACursorPos: TPoint;
   const ACursor: HCURSOR;
   const AStream: Classes.TStream;
   const A8Bit: Boolean = True
); overload;
```

Parameters

ARect [in]
Screen area to take screenshot. Typically, this is a return value from the FindScreenshotRect 400 function.

AFocusWnd [in, optional]
A window which borders should be highlighted. It does not have to be a top level window. This is typically a child control (such as a button), which indicates focused control. Pass 0 to disable this feature.

ACursorPos [in]
Absolute screen coordinates of mouse cursor. This is used to include image of mouse cursor in the screenshot.

ACursor [in]
Mouse cursor image. This is used to include image of mouse cursor in the screenshot.

AFileName [in]
Full absolute file name to save screenshot to.

A8Bit [in, optional]
True (default) - use 256-color image (to save space), False - use 32-bit color image.

AStream [in]
Data stream to save screenshot to.

Remarks

This routine takes screenshot with specified parameters and saves it to file or data stream.

Note
Effectively, this routines is a simple combination of ScreenshotInit 543 + ScreenshotSave 544 + ScreenshotFree 543.

See also

- FindScreenshotRect Routine 400
- ScreenshotInit Routine 543

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.265 ScreenshotFree

Release screenshot.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
procedure ScreenshotFree(
   const AScreenshot: TScreeenshotHandle 721
);
```

Parameters

AScreenshot [in]

Initialized screenshot. This is a return value from the ScreenshotInit 543 function.

Remarks

This routine releases screenshot created by the ScreenshotInit 543 routine.

After a call to this function: `AScreenshot` is no longer valid and must not be used.

See also

- ScreenshotInit Routine 543

10.266 ScreenshotInit

Creates screenshot.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
function ScreenshotInit(
   const ARect: TRect;
   const AFocusWnd: HWND;
   const ACursorPos: TPoint;
   const ACursor: HCURSOR
): TScreeenshotHandle 721;
```

Parameters

ARect [in]

Screen area to take screenshot. Typically, this is a return value from the FindScreenshotRect 400 function.

AFocusWnd [in, optional]

A window which borders should be highlighted. It does not have to be a top level window. This is typically a child control (such as a button), which indicates focused control. Pass 0 to disable this feature.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ACursorPos [in]
Absolute screen coordinates of mouse cursor. This is used to include image of mouse cursor in the screenshot.

ACursor [in]
Mouse cursor image. This is used to include image of mouse cursor in the screenshot.

Return value

Descriptor of the created screenshot. This can be passed to the ScreenshotSave 544 function. It must be released by a call to the ScreenshotFree 543 function.

Remarks

This function initializes screenshot with the specified parameters.

The screenshot descriptor must be released by a call to the ScreenshotFree 543 function.

Note
This is a low-level function. If you need to simply take a screenshot - use Screenshot 541 function.

See also

- FindScreenshotRect Routine 400
- ScreenshotSave Routine 544
- ScreenshotFree Routine 543
- Screenshot Routine 541

10.267 ScreenshotSave

Saves screenshot to file or stream.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
procedure ScreenshotSave(
   const AScreenshot: TScreeenshotHandle 721;
   const AFileName: String;
   const A8Bit: Boolean = True
); overload;

procedure ScreenshotSave(
   const AScreenshot: TScreeenshotHandle 721;
   const AStream: Classes.TStream;
   const A8Bit: Boolean = True
); overload;
```

Parameters

AScreenshot [in]
Initialized screenshot. This is a return value from the ScreenshotInit 543 function.

AFileName [in]
Full absolute file name to save screenshot to.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

A8Bit [in, optional]
True (default) - use 256-color image (to save space), False - use 32-bit color image.

AStream [in]
Data stream to save screenshot to.

Remarks

This routine saves screenshot which was prepared by a call to the ScreenshotInit 543 routine.

Screenshot can be saved to file or data stream.

See also

- ScreenshotInit Routine 543
- ScreenshotFree Routine 543

10.268 SDBMHash

Calculates SDBM hash.

Unit

EHash 97

Syntax

Code (Delphi)

```
function SDBMHash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSDBMHash 722; overload;

function SDBMHash(
   const AData: RawByteString
): TSDBMHash 722; overload;
```

Parameters

AData [in]
A data to calculate hash.

ADataSize [in]
Size of `AData` in bytes.

Return value

SDBM hash of the specified `AData`.

Remarks

SDBM hash is good general hashing function with good distribution. It was found to do well in scrambling bits, causing better distribution of the keys and fewer splits. This is one of the algorithms used in the Berkeley DB. EurekaLog uses SDBM as general-purpose hash for hashed lists.

10.269 SecureFree

Wipes and releases memory.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure SecureFree(
   var P: Pointer;
   const ADataSize: Cardinal
); overload;

procedure SecureFree(
   var Str: String
); overload;

procedure SecureFree(
   var Str: RawByteString
); overload;

procedure SecureFree(
   var AStream: Classes.TMemoryStream
); overload;

procedure SecureFree(
   var AStream: Classes.TCustomMemoryStream
); overload;

procedure SecureFree(
   var AList: TStringList
); overload;

procedure SecureFree(
   var IV: TTwofishInitVector 739
); overload;

procedure SecureFree(
   var AKey: TTwofishRAWKey 740
); overload;

procedure SecureFree(
   var AKey: TTEAKey 737
); overload;

procedure SecureFree(
   var AKey: TRSAKey 719
); overload;

procedure SecureFree(
   var ABuffer: TEncryptBuffer 671
); overload;
```

Remarks

Set of `SecureFree` functions takes different arguments, erases memory's content with zeros, then finally releases the memory. It is an analog of `ZeroMemory` (or `FillChar`) + `FreeMem`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.270 SecureFree

Unit
EHash 97

Syntax

Code (Delphi)

```
procedure SecureFree(
   var AHash: TCRC16Hash 621
); overload;

procedure SecureFree(
   var AHash: TCRC32Hash 622
); overload;

procedure SecureFree(
   var AHash: TMD5Hash 705
); overload;

procedure SecureFree(
   var AHash: TSHA1Hash 730
); overload;

procedure SecureFree(
   var AHash: TSHA256Hash 730
); overload;

procedure SecureFree(
   var AHash: TSDBMHash 722
); overload;
```

Remarks

10.271 SetAppType

Set a type of this application.

Unit
EAppType 54

Syntax

Code (Delphi)

```
procedure SetAppType(
   const AAppType: TAppType 610
);
```

Parameters
AAppType

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Type of application to set. This is one of `EApplicationTypes.atXXX` constants. Like `atVCL`, `atConsole` and so on.

Remarks

> **Tip**
>
> This is routine from low-level unit. Do not use this unit unless you need to extend or alter EurekaLog behavior.

Examples of application's type are: VCL, CLX, VCLX, console, service, CGI, IntraWeb and so on.

An application type is arbitrary non-empty string.

Application may have more than one type in some cases. For example, it can be both console and CGI.

Application type is generic qualifier that affects EurekaLog behaviour. For example, VCL application intercepts Application.ShowException and uses window error dialog.

EurekaLog plugins can call this routine to indicate that they're installed. For example, EAppVCL calls this as `SetAppType(atVCL)` - indicating that application is VCL application.

You can use IsAppType 442 function to check application for being of a known type.

There are also simple wrappers for more easy access. For example, IsVCL 469, IsConsole 447 and so on. See EAppType 54 unit for more info.

See also

- IsAppType 442

10.272 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

ExceptionLog7 141

Syntax

> **Code (Delphi)**

```
function SetEurekaLogState(
   const AActivate: Boolean
): Boolean;
```

Parameters

AActivate [in]

`True` - if you want to enable EurekaLog, `False` - if you want to disable EurekaLog.

Return value

Previous EurekaLog state: True if EurekaLog was active, False - otherwise.

Remarks

You can use this procedure to manually enable and disable EurekaLog in your application.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EurekaLog is considered to be active if it should do its work. You can get EurekaLog activation status by calling IsEurekaLogActive 3 procedure.

If EurekaLog is not installed - then it's always disabled (not active).

This function changes global status for all application. If you want to change activation status on per-thread basis - use SetEurekaLogStateInThread 7 procedure.

Note
This is a wrapper for EBase.SetEurekaLogState 549 function.

Examples

Code (Delphi)

```
// Enable EurekaLog in application (by default)
SetEurekaLogState(True);
raise Exception.Create('Error message'); // <- this exception will be ha
```

Code (Delphi)

```
// Disable EurekaLog in application
SetEurekaLogState(False);
raise Exception.Create('Error message'); // <- this exception will not b
```

See also

- IsEurekaLogActive 3
- SetEurekaLogStateInThread 7

10.273 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

EBase 57

Syntax

Code (Delphi)

```
function SetEurekaLogState(
   const AActivate: Boolean
): Boolean;
```

Parameters

AActivate [in]

`True` - if you want to enable EurekaLog, `False` - if you want to disable EurekaLog.

Remarks

You can use this procedure to manually enable and disable EurekaLog in your application.

EurekaLog is considered to be active if it should do its work. You can get EurekaLog activation status by calling IsEurekaLogActive 452 procedure.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

If EurekaLog is not installed - then it's always disabled (not active).

This function changes global status for all application. If you want to change activation status on per-thread basis - use SetEurekaLogStateInThread 7 procedure.

Examples

Code (Delphi)

```
// Enable EurekaLog in application (by default)
SetEurekaLogState(True);
raise Exception.Create('Error message'); // <- this exception will be ha
```

Code (Delphi)

```
// Disable EurekaLog in application
SetEurekaLogState(False);
raise Exception.Create('Error message'); // <- this exception will not b
```

See also

- IsEurekaLogActive 452
- _IsEurekaLogActive
- SetEurekaLogStateInThread 7

10.274 SetEurekaLogStateInThread

Enables or disables EurekaLog in the specified thread.

Unit

ExceptionLog7 141

Syntax

Code (Delphi)

```
function SetEurekaLogStateInThread(
   const AThreadID: Cardinal;
   const AActivate: Boolean
): Boolean;
```

Parameters

AThreadID [in, optional]

TID - thread ID which EurekaLog's status should be checked. Use 0 (default) for current thread.

AActivate [in]

`True` - if you want to enable EurekaLog, `False` - if you want to disable EurekaLog.

Return value

Previous EurekaLog state: True if EurekaLog was active, False - otherwise.

Remarks

You can use this procedure to manually enable and disable EurekaLog in the specified thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EurekaLog is considered to be active if it should do its work. You can get per-thread EurekaLog activation status by calling IsEurekaLogActiveInThread 5 procedure.

This function changes per-thread activation status. If you want to change global activation status - use SetEurekaLogState 6 procedure.

Note
This is a wrapper for EThreadsManager.SetEurekaLogStateInThread function.

Examples

Code (Delphi)

```
// Enable EurekaLog in current thread (by default)
SetEurekaLogStateInThread(True);
raise Exception.Create('Error message'); // <- this exception will be ha
```

Code (Delphi)

```
// Disable EurekaLog in current thread
SetEurekaLogStateInThread(False);
raise Exception.Create('Error message'); // <- this exception will not b
```

See also

- IsEurekaLogActive 3
- SetEurekaLogState 6

10.275 SetExceptionMessage

Sets or changes error message for an exception.

Unit

EBase 57

Syntax

Code (Delphi)

```
procedure SetExceptionMessage(
   const E: TObject;
   const AMessage: String
);
```

Parameters

E [in]
Exception object to change its `.Message` property.

AMessage [in]
An error message to assign to the specified exception object.

Remarks

EurekaLog captures exception's information when EurekaLog first encounters it. If you change exception's properties manually later - EurekaLog have no way of knowing that. This function allows you to set

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

`.Message` property of the exception object and notify EurekaLog about this change.

Examples

Code (Delphi)

```
try
   // some code
except
   on E: ESomeException do
   begin
      SetExceptionMessage(E,
         E.Message + sLineBreak + 'Some additional information');
      raise;
   end;
end;
```

See also

- ExceptionMessage Function 397

10.276 SetupConfig

Changes EurekaLog configuration storage.

Unit

EConfig 63

Syntax

Code (Delphi)

```
procedure SetupConfig(
   const AUseRegistry: Boolean;
   const AKeyPathOrFileName: String = ''
);
```

Parameters

AUseRegistry

True - use registry, False - use .ini-file.

AKeyPathOrFileName

If AUseRegistry is True then this parameter specifies registry key to store configuration.

If AUseRegistry is False then this parameter specifies .ini-file name to store configuration.

Remarks

Allows code to change default configuration storage location.

This function should be called only once at application's startup (preferably in unit's **initialization** section).

See EConfig 63 unit description for basic desciption of EurekaLog configuration.

Examples

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
ConfigFileName := '';
if not ReadCmdLineSwitch('config', ConfigFileName) then
   ConfigFileName := ExtractFilePath(ParamStr(0)) + IniFileName;
if FileExists(ConfigFileName) then
   SetupConfig(False, ConfigFileName)
else
   SetupConfig(True, GeneralEurekaLogViewerRegistryKey);
```

See also

- EConfig 63
- ConfigReadString 338
- ConfigWriteString 342
- ConfigIsRegistry 333

10.277 SetUserEmail

Sets user's e-mail.

Unit

ESysInfo 134

Syntax

Code (Delphi)

```
procedure SetUserEmail(
   const AUserEmail: String
);
```

Parameters

AUserEmail [in]

A user e-mail to be saved in registry.

Remarks

The e-mail value is stored in the registry under EurekaLog info key. It can be retrieved later by the GetUserEMail 432 function. Usually this value is set by dialogs when user enters his e-mail for sending.

See also

- GetUserEMail 432

10.278 SHA1Hash

Calculates SHA-1 hash.

Unit

EHash 97

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function SHA1Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSHA1Hash 730; overload;

function SHA1Hash(
   const AData: RawByteString
): TSHA1Hash 730; overload;
```

Parameters

AData [in]
A data to calculate hash.

ADataSize [in]
Size of AData in bytes.

Return value

SHA-1 hash of the specified AData.

Remarks

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.

SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec. SHA-1 hashing is also used in distributed revision control systems like Git, Mercurial, and Monotone to identify revisions, and to detect data corruption or tampering.

Important!

SHA-1 is considered insecure against well-funded opponents. It is recommended to use SHA-2 or SHA-3 instead. Microsoft, Google and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017. Windows XP SP2 and earlier, and Android 2.2 and earlier, do not support SHA2 certificates.

See also

- SHA1HMAC Function 554

10.279 SHA1HMAC

Calculates HMAC-SHA1.

Unit

EHash 97

Syntax

Code (Delphi)

```
function SHA1HMAC(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const APassword: Pointer;
   const APasswordSize: Cardinal
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): TSHA1Hash; overload;

function SHA1HMAC(
   const AData, APassword: RawByteString
): TSHA1Hash; overload;
```

Parameters

AData [in]
A data to calculate value ("text", "message").

ADataSize [in]
Size of AData in bytes.

APassword [in]
A common shared secret ("password", "key").

APasswordSize [in]
Size of APassword in bytes.

Return value

SHA1 HMAC value of the specified AData and APassword.

Remarks

A Keyed-Hash Message Authentication Code (HMAC) is a specific type of Message Authentication Code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.

Any cryptographic hash function, such as MD5, SHA-1, or SHA-256, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5, HMAC-SHA1, or HMAC-SHA256 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

Important!

SHA-1 is considered insecure against well-funded opponents. It is recommended to use SHA-2 or SHA-3 instead. Microsoft, Google and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017. Windows XP SP2 and earlier, and Android 2.2 and earlier, do not support SHA2 certificates.

See also

- SHA1Hash Function
- MD5HMAC Function
- SHA256HMAC Function

10.280SHA256Hash

Calculates SHA-2/SHA-256 hash.

Unit

EHash

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
function SHA256Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSHA256Hash730; overload;

function SHA256Hash(
   const AData: RawByteString
): TSHA256Hash730; overload;
```

Parameters

AData [in]
A data to calculate hash.

ADataSize [in]
Size of AData in bytes.

Return value

SHA-256 hash of the specified AData.

Remarks

SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. SHA-256 and SHA-512 are novel hash functions computed with 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds.

The SHA-2 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

Important!
SHA-2 is considered to be more secure than SHA-1.

10.281SHA256HMAC

Calculates HMAC-SHA256.

Unit

EHash97

Syntax

Code (Delphi)

```
function SHA256HMAC(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const APassword: Pointer;
   const APasswordSize: Cardinal
): TSHA256Hash730; overload;

function SHA256HMAC(
   const AData, APassword: RawByteString
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
): TSHA256Hash730; overload;
```

Parameters

AData [in]
A data to calculate value ("text", "message").

ADataSize [in]
Size of AData in bytes.

APassword [in]
A common shared secret ("password", "key").

APasswordSize [in]
Size of APassword in bytes.

Return value

SHA256 HMAC value of the specified AData and APassword.

Remarks

A Keyed-Hash Message Authentication Code (HMAC) is a specific type of Message Authentication Code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.

Any cryptographic hash function, such as MD5, SHA-1, or SHA-256, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5, HMAC-SHA1, or HMAC-SHA256 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

See also

- SHA256Hash Function555
- MD5HMAC Function481
- SHA1HMAC Function554

10.282ShowExpected

Shows expected exception.

Unit

EBase57

Syntax

Code (Delphi)

```
procedure ShowExpected(
   const AException: TObject;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID = 0
); overload;

procedure ShowExpected(
   const AException: TObject;
   const AURL: String;
   const ABugID: TBugID = 0
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
); overload;

procedure ShowExpected(
   const AMessage: String;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID = 0
); overload;

procedure ShowExpected(
   const AMessage: String;
   const AURL: String;
   const ABugID: TBugID = 0
); overload;
```

Parameters

AException [in]
Exception object to show.

AContextNumber [in, optional]
Help context number for this exception.

ABugID [in, optional]
BugID for this exception.

AURL [in, optional]
URL for this exception.

AMessage [in]
Exception message.

Remarks

This is equal to RaiseExpected, except the exception will be immediately handled (processed). Which usually result in showing error dialog.

Note
For overloaded `ShowExpected` with `AMessage` - a `EPlannedException` object will be constructed with the specified message.

See also

- RaiseExpected Function 499
- RaiseOuterExceptionExpected Function 501
- MarkExpected Function 480
- IsExpected Function 455

10.283 SignatureGetInfo

Returns information about digital signature and its certificate of an executable file.

Unit

ESignature 131

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
function SignatureGetInfo(
   const AFileName: String
): TSignatureInfo 731;
```

Parameters

AFileName [in]

Full absolute path to executable file (.exe, .dll, .bpl). File must exists and be readable.

Return value

Information about digital signature of `AFileName`.

Remarks

This function reads the specified file and returns information about its digital signature and certificate that was used to create the signature.

See also

- TSignatureInfo Record 731

10.284 TEADecrypt

Decrypts encrypted data by using TEA cipher.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure TEADecrypt(
   const AKey: TTEAKey 737;
   const AEncryptedData, AData: Pointer;
   const ADataSize: Cardinal
); overload;

function TEADecrypt(
   const AKey: TTEAKey 737;
   const AEncryptedData: RawByteString
): RawByteString; overload;

procedure TEADecrypt(
   const AKey: TTEAKey 737;
   AInPlaceData: TEncryptBuffer 671
); overload;
```

Parameters

AKey [in]

A key for decryption from TEADeriveKey 565 function.

AEncryptedData [in]

A data to decrypt. Must not contain salt (see TEADeriveKey 565). Must be in little-endian format.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AData [out]
A decrypted AEncryptedData's "clear text" data. Must have the same size as AEncryptedData.

ADataSize [in]
Size of AEncryptedData/AData.

AEncryptedData [in]
Encrypted data (in string form) to decrypt. Must not contain salt (see TEADeriveKey 565). Must be in little-endian format.

AInPlaceData [in, out]
On input: encrypted buffer to decrypt. Must not contain salt (see TEADeriveKey 565). Must be in little-endian format.
On output: "clear text" buffer (same size).

Return value

Decrypted "clear text" data.

Remarks

This function decrypts data previosly encrypted with TEAEncrypt 572 function by symmetric TEA cipher. The encrypted data is passed in AEncryptedData parameter. The function outputs decrypted data back to AData parameter. AData expected to have the same size as AEncryptedData. In other words, symmetric decryption with TEA does not change the size of the data.

Note
AData value can also point to the same place as AEncryptedData - to make "in place" decryption, but it is better to use overloaded variant of TEADecrypt with AInPlace parameter.

The overload variant with RawByteString decrypts AEncryptedData parameter and outputs decrypted data as function's result.

The overload variant with AInPlaceData decrypts data "in place".

Tip
Using salt in TEADeriveKey 565 function means that final size of encrypted/decrypted data **will** change - because salt must be stored with encrypted data. If you do not use salt - the data size will be the same. However, data size for TEADecrypt function is never changed (even if salt is used).

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key.

Important!
TEA has a few weaknesses. Most notably, it suffers from equivalent keys—each key is equivalent to three others, which means that the effective key size is only 126 bits. As a result, TEA is especially bad as a cryptographic hash function. TEA is also susceptible to a related-key attack which requires 2^23 chosen plaintexts under a related-key pair, with 2^32 time complexity.

Examples

Code (Delphi)
var

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   Key: TTEAKey;
   MS: TMemoryStream;
   Data: Pointer;
   DataSize: Integer;
   FS: TFileStream;
begin
   MS := TMemoryStream.Create;
   try
      // Load C:\Text.bin file into MS
      MS.LoadFromFile('C:\Text.bin');

      // Allocate buffer for decryption - must have the same size
      DataSize := MS.Size;
      Data := AllocMem(DataSize);
      try

         // Create new key from password
         Key := TEADeriveKey('password'); // - always the same for the s

         // Decrypt MS into Data
         TEADecrypt(Key, MS.Memory, Data, MS.Size { same as DataSize } )

         // Wipe key from memory - do this as soon as possible
         TEADestroyKey(Key); // - same as SecureFree(Key);

         // No longer needed, so we can safely delete it
         SecureFree(MS);

         // Save decrypted buffer into C:\Text.txt
         FS := TFileStream.Create('C:\Text.txt', fmCreate or fmShareExcl
         try
          FS.WriteBuffer(Data^, DataSize);
         finally
          FreeAndNil(FS);
         end;

      finally
         SecureFree(Data, DataSize);
      end;

   finally
      FreeAndNil(MS);
   end;
end;
```

Code (Delphi)

```
var
   Key: TTEAKey;
   EncryptedText: RawByteString;
   S: String;
begin
   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will be the same for the same original text and pa
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa

   // Decrypt text back
   // Use encrypted RAW binary, not the hex-converted text
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

Code (Delphi)

```
var
   Key: TTEAKey;
   EncryptedData: TEncryptBuffer;
   FS: TFileStream;
begin
   // Mark buffer as emtpy
   EncryptedData.pbData := nil;
   EncryptedData.cbData := 0;
   try

      // Load C:\Text.bin file into EncryptedData
      FS := TFileStream.Create('C:\Text.bin', fmOpenRead or fmShareDenyW
      try
         EncryptedData.cbData := FS.Size;
         EncryptedData.pbData := AllocMem(EncryptedData.cbData);
         FS.ReadBuffer(EncryptedData.pbData^, EncryptedData.cbData);
      finally
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
         FreeAndNil(FS);
      end;

      // Create new key from password
      Key := TEADeriveKey('password'); // - always the same for the same

      // Decrypt EncryptedData in place
      TEADecrypt(Key, EncryptedData);

      // Wipe key from memory - do this as soon as possible
      TEADestroyKey(Key); // - same as SecureFree(Key);

      // EncryptedData now holds unencrypted "clear text" data

      // Save "clear text" data into new file
      FS := TFileStream.Create('C:\Text.txt', fmCreate or fmShareExclusi
      try
         FS.WriteBuffer(EncryptedData.pbData^, EncryptedData.cbData);
      finally
         FreeAndNil(FS);
      end;

   finally
      SecureFree(EncryptedData);
   end;
end;
```

See also

- TEADeriveKey Function 565
- TEAEncrypt Function 572

10.285 TEADefaultKey

Returns pre-initialized default key for TEA.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TEADefaultKey: TTEAKey 737;
```

Return value

Key derived from empty password.

Remarks

This function returns the same value as `TEADeriveKey('', nil)`.

Unlike TEADeriveKey 565 function, `TEADefaultKey` does not perform any calculations and simply uses pre-initialized value. Thus, you should use this function to improve performance.

Note

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Encrypting data with empty password does not mean that data will be unencrypted (e.g. not changed). It will be encrypted, but anyone will be able to decrypt it back, as password is empty. Salt is not needed for the same reason (as password is known).

Tip

Since default password is always stored in memory - there is not much meaning to wipe default key (with TEADestroyKey 572) after using it.

Examples

Code (Delphi)

```
var
   Key: TTEAKey;
   EncryptedText: RawByteString;
   S: String;
begin
   // Use default key
   Key := TEADefaultKey; // - same as TEADeriveKey('', nil);

   // Encrypt some text with the default key
   // (meaning "encrypt some text with empty password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // There is no much meaning in wiping key from memory, as it is alway

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will be the same for the same text
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Use default key
   Key := TEADefaultKey; // - same as TEADeriveKey('', nil);

   // Decrypt text back
   // Use encrypted RAW binary, not the hex-converted text
   S := TEADecrypt(Key, EncryptedText);

   // There is no much meaning in wiping key from memory, as it is alway

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEADeriveKey Function 565

10.286 TEADeriveKey

Initializes TEA key from password.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TEADeriveKey(
   const APassword: String;
   const ASalt: PSalt 609 = nil
): TTEAKey 737; overload;

function TEADeriveKey(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASalt: PSalt 609 = nil
): TTEAKey 737; overload;
```

Parameters

APassword [in, opt]
User password.

ASalt [in, opt]
Optional salt value. It is recommended to always use/specify salt when possible - to defend against dictionary and pre-computed rainbow table attacks. Use InitSalt 438 function to create new salt for encryption and store salt with encrypted data. Use stored salt to decrypt data.

AData [in]
Pointer to binary password to initialize the key.

ADataSize [in]
Size of `AData` in bytes.

Return value

TEA key derived from `APassword` and (optionally) `ASalt`.

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

This value is always the same for the same `APassword` (and same `ASalt`).

Remarks

This function converts `APassword` to UTF-8, attaches `ASalt` (if specified), then uses the resulting RAW binary string to initialize the key. This function uses `MD5(ASalt + APassword)` to derive key.

You do not need to call this function if you already have RAW binary TEA key (TTEAKey 737).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

Overloaded variant lets you initialize key from any binary password. It is usually used in conjunction with asymmetric cipher when generating random passwords for symmetric cipher.

Warning

`TEADeriveKey` and `TEADeriveKeyHMAC` use different algorithms to derive the key. E.g. key created by the `TEADeriveKey` will be different from the key created by the `TEADeriveKeyHMAC` function. In other words, if you use some function on sending side - then you **must** use the same function on receiving side. If you do not have compatibility requirements with external code - we recommend to use `TEADeriveKeyHMAC` function.

Tip

`APassword` can also be an empty string. Encrypting data with empty password does not mean that data will be unencrypted (e.g. not changed). It will be encrypted, but anyone will be able to decrypt it back, as password is empty. It is strongly recommended not to use empty passwords. There is no need to specify salt for empty password. It is better to use TEADefaultKey 563 function in this case.

Important!

Empty salt (zeroed) is not the same as no salt. Use InitSalt 438 function to create new random salt. Once created, salt **must** be stored with encrypted data and then passed to `TEADeriveKey` for decryption.

Tip

Using salt means that size of encrypted/decrypted data **will** change - because salt must be stored with encrypted data. If you do not use salt - the data size will not change during encryption/decryption.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key.

Important!

TEA has a few weaknesses. Most notably, it suffers from equivalent keys—each key is equivalent to three others, which means that the effective key size is only 126 bits. As a result, TEA is especially bad as a cryptographic hash function. TEA is also susceptible to a related-key attack which requires 2^23 chosen plaintexts under a related-key pair, with 2^32 time complexity.

Examples

Code (Delphi)

```
var
   Key: TTEAKey;
   EncryptedText: RawByteString;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   S: String;
begin
   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa

   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will be the same for the same original text and pa
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa

   // Decrypt text back
   // Use encrypted RAW binary, not the hex-converted text
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

Code (Delphi)

```
var
   Key: TTEAKey;
   Salt: TSalt;
   EncryptedText: RawByteString;
   S: String;
   SaltEncodedLen: Integer;
begin
   // Create new (random) salt
   Salt := InitSalt;

   // Create new key from password
   // Key will be different each time
   // Because salt is different
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   Key := TEADeriveKey('password', @Salt);

   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key/salt from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // You must store salt with encrypted data
   EncryptedText :=
      RAWToString(@Salt, SizeOf(Salt)) +
      EncryptedText;

   // No longer needed, erase it
   SecureFree(Salt);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will always be different
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Read stored salt
   SaltEncodedLen := HexCalcEncodedSize(SizeOf(Salt)));
   RAWFromString(Copy(EncryptedText, 1, SaltEncodedLen), @Salt);
   // Remove salt from EncryptedText
   EncryptedText := Copy(EncryptedText, SaltEncodedLen + 1, MaxInt);

   // Create key from password
   // Key will be the same as the one used for encryption above
   Key := TEADeriveKey('password', @Salt);

   // No longer needed, erase it
   SecureFree(Salt);

   // Decrypt text back
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEADeriveKeyHMAC Function 569
- TEADefaultKey Function 563
- TEAInitSessionKey Function 576
- InitSalt Function 438
- TEAEncrypt Function 572
- TEADecrypt Function 559
- TEADestroyKey Function 572
- MD5Hash Function 480

10.287 TEADeriveKeyHMAC

Initializes TEA key from password via MD5 HMAC.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TEADeriveKeyHMAC(
   const APassword: String;
   const ASalt: TSalt 721
): TTEAKey 737; overload;
```

Parameters

APassword [in]

User password.

ASalt [in]

Salt value. Use InitSalt 438 function to create new salt for encryption and store salt with encrypted data. Use stored salt to decrypt data.

Return value

TEA key derived from `APassword` and `ASalt` using MD5 HMAC.

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

This value is always the same for the same `APassword` and same `ASalt`.

Remarks

This function converts `APassword` to UTF-8 and then uses MD5HMAC(ASalt, APassword) to derive the key using the specified salt.

You do not need to call this function if you already have RAW binary TEA key (TTEAKey 737).

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

Warning

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

`TEADeriveKey` and `TEADeriveKeyHMAC` use different algorithms to derive the key. E.g. key created by the `TEADeriveKey` will be different from the key created by the `TEADeriveKeyHMAC` function. In other words, if you use some function on sending side - then you **must** use the same function on receiving side. If you do not have compatibility requirements with external code - we recommend to use `TEADeriveKeyHMAC` function.

Tip

Using salt means that size of encrypted/decrypted data **will** change - because salt must be stored with encrypted data. If you do not use salt - the data size will not change during encryption/decryption.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key.

Important!

TEA has a few weaknesses. Most notably, it suffers from equivalent keys—each key is equivalent to three others, which means that the effective key size is only 126 bits. As a result, TEA is especially bad as a cryptographic hash function. TEA is also susceptible to a related-key attack which requires 2^23 chosen plaintexts under a related-key pair, with 2^32 time complexity.

Examples

Code (Delphi)

```
var
   Key: TTEAKey;
   Salt: TSalt;
   EncryptedText: RawByteString;
   S: String;
   SaltEncodedLen: Integer;
begin
   // Create new (random) salt
   Salt := InitSalt;

   // Create new key from password
   // Key will be different each time
   // Because salt is different
   Key := TEADeriveKeyHMAC('password', Salt);

   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key/salt from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // You must store salt with encrypted data
   EncryptedText :=
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
      RAWToString(@Salt, SizeOf(Salt)) +
      EncryptedText;

   // No longer needed, erase it
   SecureFree(Salt);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will always be different
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Read stored salt
   SaltEncodedLen := HexCalcEncodedSize(SizeOf(Salt)));
   RAWFromString(Copy(EncryptedText, 1, SaltEncodedLen), @Salt);
   // Remove salt from EncryptedText
   EncryptedText := Copy(EncryptedText, SaltEncodedLen + 1, MaxInt);

   // Create key from password
   // Key will be the same as the one used for encryption above
   Key := TEADeriveKeyHMAC('password', Salt);

   // No longer needed, erase it
   SecureFree(Salt);

   // Decrypt text back
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

See also

- TEADeriveKey Function 565
- TEADefaultKey Function 563
- TEAInitSessionKey Function 576
- InitSalt Function 438
- TEAEncrypt Function 572
- TEADecrypt Function 559
- TEADestroyKey Function 572
- MD5HMAC Function 481

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.288 TEADestroyKey

Fills TEA key with zeros.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure TEADestroyKey(
   var AKey: TTEAKey 737
);
```

Parameters

AKey [in]
TEA key to be wiped.

Remarks

While calling this function is not strictly required (as TEA key does not hold any allocated resources which would require cleanup), but we strongly recommend to call this function for every TEA key once you have finished working with it.

Tip

Since default password (returned by TEADefaultKey 563) is always stored in memory - there is not much meaning to wipe default key after using it.

See also

- TEADeriveKey Function 565

10.289 TEAEncrypt

Encrypts "clear text" data by using TEA cipher.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure TEAEncrypt(
   const AKey: TTEAKey 737;
   const AData, AEncryptedData: Pointer;
   const ADataSize: Cardinal
); overload;

function TEAEncrypt(
   const AKey: TTEAKey 737;
   const AData: RawByteString
): RawByteString; overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure TEAEncrypt(
   const AKey: TTEAKey 737;
   AInPlaceData: TEncryptBuffer 671
); overload;
```

Parameters

AKey [in]
A key for encryption from TEADeriveKey 565 function.

AData [in]
A data to encrypt.

AEncryptedData [out]
An encrypted AData's data. Must have the same size as AData. Data is returned in little-endian format.

ADataSize [in]
Size of AData/AEncryptedData.

AData [in]
String "clear text" data to encrypt.

AInPlaceData [in, out]
On input: "clear text" buffer to encrypt.
On output: encrypted buffer (same size, lettle-endian).

Return value

Encrypted AData's data in a string form. Data is returned in little-endian format.

Remarks

This function encrypts data passed in AData parameter by symmetric TEA cipher and outputs encrypted data back to AEncryptedData parameter. AEncryptedData expected to have the same size as AData. In other words, symmetric encryption with TEA does not change the size of the data.

Note
AEncryptedData value can also point to the same place as AData - to make "in place" encryption, but it is better to use overloaded variant of TEAEncrypt with AInPlace parameter.

The overload variant with RawByteString encrypts AData parameter and outputs encrypted data as function's result.

The overload variant with AInPlaceData encrypts data "in place".

Tip
Using salt in TEADeriveKey 565 function means that final size of encrypted/decrypted data **will** change - because salt must be stored with encrypted data. If you do not use salt - the data size will be the same. However, data size for TEAEncrypt function is never changed (even if salt is used).

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key.

Important!

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TEA has a few weaknesses. Most notably, it suffers from equivalent keys—each key is equivalent to three others, which means that the effective key size is only 126 bits. As a result, TEA is especially bad as a cryptographic hash function. TEA is also susceptible to a related-key attack which requires 2^23 chosen plaintexts under a related-key pair, with 2^32 time complexity.

Examples

Code (Delphi)

```
var
   Key: TTEAKey;
   MS: TMemoryStream;
   EncryptedData: Pointer;
   EncryptedDataSize: Integer;
   FS: TFileStream;
begin
   MS := TMemoryStream.Create;
   try
      // Load C:\Text.txt file into MS
      MS.LoadFromFile('C:\Text.txt');

      // Allocate buffer for encryption - must have the same size
      EncryptedDataSize := MS.Size;
      EncryptedData := AllocMem(EncryptedDataSize);
      try

         // Create new key from password
         Key := TEADeriveKey('password'); // - always the same for the s

         // Encrypt MS into EncryptedData
         TEAEncrypt(Key, MS.Memory, EncryptedData, MS.Size { same as Enc

         // Wipe key from memory - do this as soon as possible
         TEADestroyKey(Key); // - same as SecureFree(Key);

         // No longer needed, so we can safely delete it
         SecureFree(MS);

         // Save encrypted buffer into C:\Text.bin
         FS := TFileStream.Create('C:\Text.bin', fmCreate or fmShareExcl
         try
          FS.WriteBuffer(EncryptedData^, EncryptedDataSize);
         finally
          FreeAndNil(FS);
         end;

      finally
         SecureFree(EncryptedData, EncryptedDataSize);
      end;

   finally
      FreeAndNil(MS);
   end;
end;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
var
   Key: TTEAKey;
   EncryptedText: RawByteString;
   S: String;
begin
   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa

   // Encrypt some text with the above key
   // (meaning "encrypt some text with the above password")
   EncryptedText := TEAEncrypt(Key, 'clear text');

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // EncryptedText contains encrypted RAW bytes
   // Do not output EncryptedText to any place which expects human-reada

   // Debug output
   // Encrypted text is RAW bytes, so we need to convert it to something
   // Encrypted value will be the same for the same original text and pa
   Edit1.Text := HexEncodeString(EncryptedText);

   // _____________________________________________

   // Create new key from password
   Key := TEADeriveKey('password'); // - always the same for the same pa

   // Decrypt text back
   // Use encrypted RAW binary, not the hex-converted text
   S := TEADecrypt(Key, EncryptedText);

   // Wipe key from memory - do this as soon as possible
   TEADestroyKey(Key); // - same as SecureFree(Key);

   // Wipe also all data
   SecureFree(EncryptedText);

   // Now S = 'clear text'

   // Call SecureFree for all data once you have finished working with i
   SecureFree(S);
end;
```

Code (Delphi)

```
var
   Key: TTEAKey;
   Data: TEncryptBuffer;
   FS: TFileStream;
begin
   // Mark buffer as emtpy
   Data.pbData := nil;
   Data.cbData := 0;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   try

      // Load C:\Text.txt file into Data
      FS := TFileStream.Create('C:\Text.txt', fmOpenRead or fmShareDenyW
      try
         Data.cbData := FS.Size;
         Data.pbData := AllocMem(Data.cbData);
         FS.ReadBuffer(Data.pbData^, Data.cbData);
      finally
         FreeAndNil(FS);
      end;

      // Create new key from password
      Key := TEADeriveKey('password'); // - always the same for the same

      // Encrypt Data in place
      TEAEncrypt(Key, Data);

      // Wipe key from memory - do this as soon as possible
      TEADestroyKey(Key); // - same as SecureFree(Key);

      // Data now holds encrypted binary data

      // Debug output
      Memo1.Lines.Text := HexEncodeToString(Data.pbData, Data.cbData);

      // Save encrypted data into new file
      FS := TFileStream.Create('C:\Text.bin', fmCreate or fmShareExclusi
      try
         FS.WriteBuffer(Data.pbData^, Data.cbData);
      finally
         FreeAndNil(FS);
      end;

   finally
      SecureFree(Data);
   end;
end;
```

See also

- TEADeriveKey Function 565
- TEADecrypt Function 559

10.290 TEAInitSessionKey

Generates random TEA key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TEAInitSessionKey: TTEAKey 737;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Return value

Random TEA key.

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

This value is never the same.

Remarks

This function generates random TEA key. This key can be used as session key for asymmetric encryption. See description of RSAEncrypt 528 for more details.

Important!
Do not use RTL's Random function to create random TEA key manually. RTL implementation is a simple pseudo-random generator and it does not generate cryptographically-strong random data.

Use TEADeriveKey 565 function to derive symmetric TEA key from string password or any other arbitrary "seed" data.

See also

- RSAEncrypt Function 528

10.291 ThreadData

Returns thread information (current thread only).

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
function ThreadData: TEurekaThreadData 219;
```

Return value

Information about current thread. Do not destroy the returned object. This value could be nil for rare cases (such as calling this function before/after EThreadsManager's initialization/finalization).

Remarks

Use this function to get information about current thread.

There is no need to release the returned object.

There is no need to use locking - because informational object 219 for current thread will exist until thread is terminated. Most thread properties are read-only and don't require synchronization. Dynamic properties (such as Name, LastException, LastHTMLPage) should be modified by current thread only.

This function can return information only about current thread. Use AcquireThreadData 318 or ThreadDataEnum 578 functions to get information about arbitrary thread.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- AcquireThreadData Function 318
- ThreadDataEnum Function 578
- TEurekaThreadData Class 219

10.292 ThreadDataEnum

Enumerates running threads.

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
function ThreadDataEnum(
   const AForceAllActive: Boolean = False
): TEurekaThreadDataList 229;
```

Parameters

AForceAllActive [in, optional]

`False` (default) - return a copy of internal thread list. This list will contain all known threads. Note that known threads may not represent all threads in your application (for example, if you did not intall low-level hooks or disable EurekaLog in system threads). `True` - build and return a list of all running threads in application. Only active (still running) threads are returned.

Return value

A list of threads. You have to delete this object after use. A new object is created for each function's call.

Remarks

This function creates new list of threads and fills it with information about all known threads. The returned list should be deleted after use.

There is no need to use locking - because informational object 229 for the thread list will be individual for each function's call. Any change for dynamic properties (such as Name, LastException, LastHTMLPage) will be ignored. Use ThreadData 577 function to modify thread's dynamic properties.

See also

- ThreadData Function 577
- AcquireThreadData Function 318
- TEurekaThreadData Class 219

10.293 TwofishDecrypt

Decrypts encrypted data by using Twofish cipher.

Unit

EEncrypt 80

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure TwofishDecrypt(
   const AKey: TTwofishKey 740;
   const AEncryptedData: Pointer;
   out AData: Pointer;
   var ADataSize: Cardinal;
   const AInitVector: PTwofishInitVector 610 = nil
); overload;

function TwofishDecrypt(
   const AKey: TTwofishKey 740;
   const AEncryptedData: RawByteString;
   const AInitVector: PTwofishInitVector 610 = nil
): RawByteString; overload;
```

Parameters

AKey [in]
A key for decryption. Comes from TwofishDeriveKey 580 function or TwofishInitKey 586 function.

AEncryptedData [in]
A data to decrypt. Must be in little-endian format.

AData [out]
A decrypted AEncryptedData's "clear text" data. Must be deleted with FreeMem function (but we recommend to use SecureFree 545 function instead). Will have different size from AData. PKCS#5 padding will be removed.

ADataSize [in, out]
On input: size of AEncryptedData.
On output: size of AData.

AInitVector [in, opt]
An optional initialization vector (IV). This value **must** be the same as the one passed to TwofishEncrypt 583 function during encryption.

AEncryptedData [in]
Encrypted data (in string form) to decrypt. Must not contain salt (see TwofishDeriveKey 580). Must be in little-endian format.

Return value

Decrypted "clear text" data. PKCS#5 padding will be removed.

Remarks

This function decrypts data previosly encrypted with TwofishEncrypt 583 function with key of symmetric Twofish cipher. The encrypted data is passed in AEncryptedData parameter. The function outputs decrypted data back to AData parameter. AData expected to have different size from AEncryptedData. In other words, symmetric decryption with Twofish does change the size of the data.

The overload variant with RawByteString decrypts AEncryptedData parameter and outputs decrypted data as function's result.

Important!

Zero-filled IV is not the same as not specified IV. Not specified IV means using ECB mode. Specified IV means using CBC mode (even if IV is empty/zero filled).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Tip

`TwofishEncrypt/TwofishDecrypt` use PKCS#5 filling scheme for both ECB and CBC modes. That is: data size will be increased (during encryption) to closest block boundary (block size is 16 bytes/128 bit), the unused space (padding) will be filled with its own size (e.g. 1..16). If source data size is exactly N * 16 bytes, then one full block (16 bytes) will be added (and filled with byte = 16). On decryption data size will be simply decreased by a value specified in the last byte.

See also

- TwofishDeriveKey Function 580
- TwofishInitKey Function 586
- TwofishEncrypt Function 583

10.294 TwofishDeriveKey

Initializes Twofish key from password.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TwofishDeriveKey(
   const APassword: String;
   const ASalt: PSalt 609 = nil
): TTwofishKey 740; overload;

function TwofishDeriveKey(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASalt: PSalt 609 = nil
): TTwofishKey 740; overload;
```

Parameters

APassword [in, opt]
User password.

ASalt [in, opt]
Optional salt value. It is recommended to always use/specify salt when possible - to defend against dictionary and pre-computed rainbow table attacks. Use InitSalt 438 function to create new salt for encryption and store salt with encrypted data. Use stored salt to decrypt data.

AData [in]
Pointer to binary password to initialize the key.

ADataSize [in]
Size of `AData` in bytes.

Return value

Precalculated 256-bit Twofish key derived from `APassword` and (optionally) `ASalt` using `SHA256Hash(ASalt + APassword)`.

Delete this key by using TwofishDestroyKey 583 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

This value is always the same for the same APassword (and same ASalt).

Important!

This is not RAW key derived from password. This is precalculated array of subkeys and SBox keys for Twofish cipher. Precalculation speed ups encryption/decryption. Use TwofishInitKey 586 to precalculate key from RAW key.

Remarks

This function converts APassword to UTF-8, attaches ASalt (if specified), then uses the resulting RAW binary string to initialize the key. This function uses SHA256Hash(ASalt + APassword) to derive the key. Therefore the key has size of 256 bits.

Important!

The returned value of this function is not RAW key derived from password. The returned value is precalculated/cached array of subkeys and SBox keys for Twofish cipher. Precalculation speed ups encryption/decryption. Use TwofishInitKey 586 to precalculate key from RAW key.

Delete the returned value by using TwofishDestroyKey 583 function when you have finished working with key.

Overloaded variant lets you initialize key from any binary password.

Warning

TwofishDeriveKey and TwofishDeriveKeyHMAC use different algorithms to derive the key. E.g. key created by the TwofishDeriveKey will be different from the key created by the TwofishDeriveKeyHMAC function. In other words, if you use some function on sending side - then you **must** use the same function on receiving side. If you do not have compatibility requirements with external code - we recommend to use TwofishDeriveKeyHMAC function.

Important!

Empty salt (zeroed) is not the same as no salt. Use TwofishInitSessionKey function to create new random salt. Once created, salt **must** be stored with encrypted data and then passed to TwofishDeriveKey for decryption.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Twofish is a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. It was one of the five finalists of the Advanced Encryption Standard (AES) contest, but it was not selected for standardization. Twofish's distinctive features are the use of pre-computed key-dependent S-boxes, and a relatively complex key schedule. Twofish cipher has not been patented. As a result, the Twofish algorithm is free for anyone to use without any restrictions whatsoever. However, Twofish has seen less widespread usage than Blowfish, which has been available longer.

See also

- TwofishInitKey Function 586
- InitSalt Function 438
- TwofishEncrypt Function 583
- TwofishDecrypt Function 578

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- TwofishDestroyKey Function 583
- SHA256Hash Function 555

10.295 TwofishDeriveKeyHMAC

Initializes Twofish key from password using SHA-256 HMAC.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TwofishDeriveKeyHMAC(
   const APassword: String;
   const ASalt: TSalt 721
): TTwofishKey 740;
```

Parameters

APassword [in]
User password.

ASalt [in]
Salt value. Use InitSalt 438 function to create new salt for encryption and store salt with encrypted data. Use stored salt to decrypt data.

Return value

Precalculated 256-bit Twofish key derived from `APassword` and `ASalt` using `SHA256HMAC(ASalt, APassword)`.

Delete this key by using TwofishDestroyKey 583 function.

This value is always the same for the same `APassword` and same `ASalt`.

Important!

This is not RAW key derived from password. This is precalculated array of subkeys and SBox keys for Twofish cipher. Precalculation speed ups encryption/decryption. Use TwofishInitKey 586 to precalculate key from RAW key.

Remarks

This function converts `APassword` to UTF-8 and then uses `SHA256HMAC(ASalt, APassword)` to derive the key. Therefore the key has size of 256 bits.

Warning

`TwofishDeriveKey` and `TwofishDeriveKeyHMAC` use different algorithms to derive the key. E.g. key created by the `TwofishDeriveKey` will be different from the key created by the `TwofishDeriveKeyHMAC` function. In other words, if you use some function on sending side - then you **must** use the same function on receiving side. If you do not have compatibility requirements with external code - we recommend to use `TwofishDeriveKeyHMAC` function.

Important!

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

The returned value of this function is not RAW key derived from password/salt. The returned value is precalculated/cached array of subkeys and SBox keys for Twofish cipher. Precalculation speed ups encryption/decryption. Use TwofishInitKey 586 to precalculate key from RAW key.

Delete the returned value by using TwofishDestroyKey 583 function when you have finished working with key.

Twofish is a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. It was one of the five finalists of the Advanced Encryption Standard (AES) contest, but it was not selected for standardization. Twofish's distinctive features are the use of pre-computed key-dependent S-boxes, and a relatively complex key schedule. Twofish cipher has not been patented. As a result, the Twofish algorithm is free for anyone to use without any restrictions whatsoever. However, Twofish has seen less widespread usage than Blowfish, which has been available longer.

See also

- TwofishInitKey Function 586
- InitSalt Function 438
- TwofishEncrypt Function 583
- TwofishDecrypt Function 578
- TwofishDestroyKey Function 583
- SHA256HMAC Function 556

10.296 TwofishDestroyKey

Deletes Twofish key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
procedure TwofishDestroyKey(
   var AKey: TTwofishKey 740
);
```

Parameters

AKey [in]
Twofish key to be released.

Remarks

This function deletes Twofish key allocated by TwofishDeriveKey 580 or TwofishInitKey 586 functions.

Call this function to release Twofish key once you have finished working with it.

See also

- TwofishDeriveKey Function 580
- TwofishInitKey Function 586

10.297 TwofishEncrypt

Encrypts "clear text" data by using Twofish cipher.

Unit

EEncrypt 80

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure TwofishEncrypt(
   const AKey: TTwofishKey740;
   const AData: Pointer;
   out AEncryptedData: Pointer;
   var ADataSize: Cardinal;
   const AInitVector: PTwofishInitVector610 = nil
); overload;

function TwofishEncrypt(
   const AKey: TTwofishKey740;
   const AData: RawByteString;
   const AInitVector: PTwofishInitVector610 = nil
): RawByteString; overload;
```

Parameters

AKey [in]
A key for encryption. Comes from TwofishDeriveKey580 function or TwofishInitKey586 function.

AData [in]
A data to encrypt. Data will be padded with PKCS#5.

AEncryptedData [out]
An encrypted AData's data. Must be deleted with FreeMem function (but we recommend to use SecureFree545 function instead). Will have different size from AData. Data is returned in little-endian format.

ADataSize [in, out]
On input: size of AData.
On output: size of AEncryptedData (will be different).

AInitVector [in, opt]
An optional initialization vector (IV) from TwofishInitIV585 function.

Default is nil - which means using ECB encryption mode (not recommended).

Specified (non-nil) value means using CBC encryption mode.

AData [in]
String "clear text" data to encrypt. Data will be padded with PKCS#5.

Return value

Encrypted AData's data in a string form. Data is returned in little-endian format.

Remarks

This function encrypts data passed in AData parameter with key of symmetric Twofish cipher and outputs encrypted data back to AEncryptedData parameter. AEncryptedData expected to have different size from original AData. In other words, symmetric encryption with Twofish does change the size of the data.

The overload variant with RawByteString encrypts AData parameter and outputs encrypted data as function's result.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Important!

Zero-filled IV is not the same as not specified IV. Not specified IV means using ECB mode. Specified IV means using CBC mode (even if IV is empty/zero filled).

Tip

`TwofishEncrypt/TwofishDecrypt` use PKCS#5 filling scheme for both ECB and CBC modes. That is: data size will be increased (during encryption) to closest block boundary (block size is 16 bytes/128 bit), the unused space (padding) will be filled with its own size (e.g. 1..16). If source data size is exactly N * 16 bytes, then one full block (16 bytes) will be added (and filled with byte = 16). On decryption data size will be simply decreased by a value specified in the last byte.

See also

- TwofishDeriveKey Function 580
- TwofishInitKey Function 586
- TwofishInitIV Function 585
- TwofishDecrypt Function 578

10.298 TwofishInitIV

Creates new IV.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TwofishInitIV: TTwofishInitVector 739;
```

Return value

Random initialization vector.

Remarks

This function creates new random initialization vector (IV). Pass the returned value to TwofishEncrypt 583 function to use CBC mode.

Important!

Do not pass the returned value to TwofishDecrypt 578 function. IV for decryption **must** match the IV used for encryption. In other words, you **must** store IV with encrypted data.

Important!

Do not use RTL's Random function to create random salt manually. RTL implementation is a simple pseudo-random generator and it does not generate cryptographically-strong random data.

Note

Initialization vector (IV) is a fixed-size input to a cryptographic primitive that is typically required to be random or pseudorandom. Randomization is crucial for encryption schemes to achieve semantic security, a property whereby repeated usage of the scheme under the same key does not allow an attacker to infer relationships between segments of the encrypted message.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TwofishEncrypt Function 583
- TwofishDecrypt Function 578

10.299 TwofishInitKey

"Imports" (precalculates) Twofish key from RAW key value.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
function TwofishInitKey(
   const AKey: TTwofishRAWKey 740
): TTwofishKey 740;
```

Parameters

AKey [in]
256-bit RAW Twofish key.

Return value

Twofish key precalculated from AKey.

Delete this key by using TwofishDestroyKey 583 function.

This value is always the same for the same AKey.

Remarks

EurekaLog uses precalculated/cached array of subkeys and SBox keys for Twofish cipher. It improves performance of encryption/decryption. Use this function to obtain precalculated key from given RAW key value.

Use TwofishDeriveKey 580 function to derive key from password.

Delete the returned value by using TwofishDestroyKey 583 function when you have finished working with key.

See also

- TwofishDeriveKey Function 580
- TwofishEncrypt Function 583
- TwofishDecrypt Function 578
- TwofishDestroyKey Function 583

10.300 TZipCompressionToString

Converts ZIP compression method value to string representation.

Unit

EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function TZipCompressionToString(
   Compression: TZipCompression 741
): string;
```

Parameters

Compression

Compression method to get its textual representation.

Return value

String representation of `Compression` argument.

Remarks

Example return values are:

- 'Unknown'
- 'Stored'
- 'Shrunk'
- 'Reduced1'
- 'Reduced2'
- 'Imploded'
- 'Tokenized'
- Etc.

10.301 UnRegisterEventAttachedFilesRequest

Unregisters previously registered OnAttachedFilesRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventAttachedFilesRequest(
   const AHandler: TELEvAttachedFilesRequestMeth 21
); overload;

procedure UnRegisterEventAttachedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvAttachedFilesRequestProc 625
); overload;
```

Parameters

AHandler [in]

Event handler to unregister.

ACustom [in, optional]

(*for procedural event handlers only*)

Custom pointer to be passed into handler.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 504

10.302UnRegisterEventBeginReportGen

Unregisters previously registered OnBeginReportGen event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventBeginReportGen(
   const AHandler: TELEvReportGenMeth 30
); overload;

procedure UnRegisterEventBeginReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 657
); overload;
```

Parameters

AHandler [in]

Event handler to unregister.

ACustom [in, optional]

(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- General events concept 83
- Registering event handler 505

10.303 UnRegisterEventCustomBugID

Unregisters previously registered OnCustomBugID event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomBugID(
   const AHandler: TELEvCustomBugIDMeth 25
); overload;

procedure UnRegisterEventCustomBugID(
   const ACustom: Pointer;
   const AHandler: TELEvCustomBugIDProc 626
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 506

10.304 UnRegisterEventCustomButtonClick

Unregisters previously registered OnCustomButtonClick event handler.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomButtonClick(
   const AHandler: TELEvCustomButtonClickMeth 28
); overload;

procedure UnRegisterEventCustomButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvCustomButtonClickProc 628
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 507

10.305 UnRegisterEventCustomDataRequest

Unregisters previously registered OnCustomDataRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomDataRequest(
   const AHandler: TELEvCustomDataRequestMeth 26
); overload;

procedure UnRegisterEventCustomDataRequest(
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const ACustom: Pointer;
   const AHandler: TELEvCustomDataRequestProc 631
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 508

10.306 UnRegisterEventCustomFileName

Unregisters previously registered OnCustomFileName event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomFileName(
   const AHandler: TELEvCustomFileNameMeth 24
); overload;

procedure UnRegisterEventCustomFileName(
   const ACustom: Pointer;
   const AHandler: TELEvCustomFileNameProc 633
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 509

10.307 UnRegisterEventCustomWebFieldsRequest

Unregisters previously registered OnCustomWebFieldRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomWebFieldsRequest(
   const AHandler: TELEvCustomWebFieldsRequestMeth 27
); overload;

procedure UnRegisterEventCustomWebFieldsRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomWebFieldsRequestProc 635
); overload;
```

Parameters

AHandler [in]

Event handler to unregister.

ACustom [in, optional]

(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 510

10.308 UnRegisterEventEndReportGen

Unregisters previously registered OnEndReportGen event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventEndReportGen(
   const AHandler: TELEvReportGenMeth 30
); overload;

procedure UnRegisterEventEndReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 657
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 511

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

10.309 UnRegisterEventExceptionAction

Unregisters previously registered OnExceptionAction event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventExceptionAction(
   const AHandler: TELEvExceptionActionMeth 16
); overload;

procedure UnRegisterEventExceptionAction(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionActionProc 638
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 512

10.310 UnRegisterEventExceptionError

Unregisters previously registered OnExceptionError event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
procedure UnRegisterEventExceptionError(
   const AHandler: TELEvExceptionErrorMeth 18
); overload;

procedure UnRegisterEventExceptionError(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionErrorProc 642
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 513

10.311 UnRegisterEventExceptionNotify

Unregisters previously registered OnExceptionNotify event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventExceptionNotify(
   const AHandler: TELEvExceptionNotifyMeth 14
); overload;

procedure UnRegisterEventExceptionNotify(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionNotifyProc 645
); overload;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 514

10.312 UnRegisterEventHelpButtonClick

Unregisters previously registered OnHelpButtonClick event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventHelpButtonClick(
   const AHandler: TELEvHelpButtonClickMeth 29
); overload;

procedure UnRegisterEventHelpButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvHelpButtonClickProc 648
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 515

10.313 UnRegisterEventPasswordRequest

Unregisters previously registered OnPasswordRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventPasswordRequest(
   const AHandler: TELEvPasswordRequestMeth 19
); overload;

procedure UnRegisterEventPasswordRequest(
   const ACustom: Pointer;
   const AHandler: TELEvPasswordRequestProc 651
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- General events concept 83
- Registering event handler 516

10.314 UnRegisterEventRaise

Unregisters previously registered OnRaise event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventRaise(
   const AHandler: TELEvRaiseMeth 12
); overload;

procedure UnRegisterEventRaise(
   const ACustom: Pointer;
   const AHandler: TELEvRaiseProc 653
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 517

10.315 UnRegisterEventReproduceRequest

Unregisters previously registered OnReproduceRequest event handler.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure UnRegisterEventReproduceRequest(
   const AHandler: TELEvReproduceRequestMeth 23
); overload;

procedure UnRegisterEventReproduceRequest(
   const ACustom: Pointer;
   const AHandler: TELEvReproduceRequestProc 659
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip

It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 518

10.316 UnRegisterEventZippedFilesRequest

Unregisters previously registered OnZippedFilesRequest event handler.

Unit

EEvents 83

Syntax

Code (Delphi)

```
procedure UnRegisterEventZippedFilesRequest(
   const AHandler: TELEvZippedFilesRequestMeth 22
); overload;

procedure UnRegisterEventZippedFilesRequest(
   const ACustom: Pointer;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const AHandler: TELEvZippedFilesRequestProc 661
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

Custom pointer to be passed into handler.

Must be the same pointer which was passed to event registration routine.

Remarks

If the specified event handler wasn't registered - does nothing. Otherwise handler is removed and it will be no longer be invoked.

Tip
It's best to call this routine in object's destructor (for method event handlers) or unit's finalization section (for procedural event handlers).

See also

- General events concept 83
- Registering event handler 519

10.317 WCTDone

Finalize EWCT unit.

Unit

EWCT 139

Syntax

Code (Delphi)

```
procedure WCTDone;
```

Remarks

This function clears all dynamically linked routines. Call this function after using EWCT unit. This is an optional action.

10.318 WCTInit

Initialize EWCT unit.

Unit

EWCT 139

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

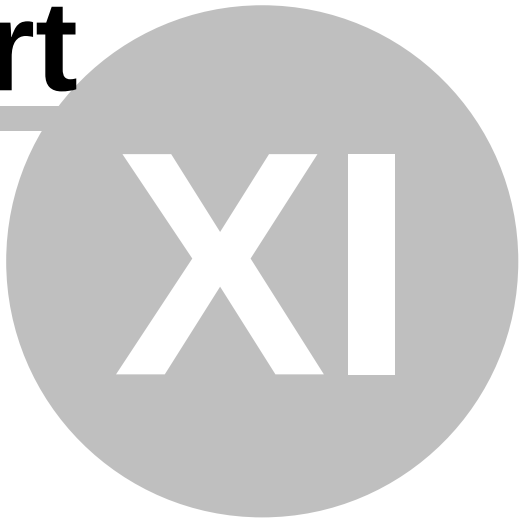
```
procedure WCTInit;
```

Remarks

This function imports all dynamically linked routines. Call this function before using EWCT unit.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part



11 All Types

PCustomName 607	Defines buffer for storing name of memory block.
PELLocationInfo 608	A pointer to code location information.
PInstruction 608	A pointer to TInstruction 687.
PPointerRecord 608	A pointer to TPointerRecord type.
PSalt 609	Pointer to salt.
PtrUInt 609	Unsigned integer type with same size as `Pointer`.
PTwofishInitVector 610	Pointer to IV for Twofish.
TAppType 610	Defines application type.
TBaseDialogClass 611	Pointer to dialog class.
TBlockType 611	Defines type of memory block.
TBugID 613	BugID type.
TCertificateInfo 613	Certificate information.
TCheckType 619	Function check kind.
TCRC16Hash 621	CRC-16 checksum.
TCRC32Hash 622	CRC-32 checksum.
TCustomName 622	Defines buffer for storing name of memory block.
TDumpCompareFunc 622	Comparator for memory blocks.
TDumpFilterFunc 623	Filter for memory blocks.
TELEvAttachedFilesRequestMeth 21	Method event handler for OnAttachedFilesRequest event.
TELEvAttachedFilesRequestProc 625	Procedural event handler for OnAttachedFilesRequest event.
TELEvCustomBugIDMeth 25	Method event handler for OnCustomBugID event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TELEvCustomBugIDProc 626	Procedural event handler for OnCustomBugID event.
TELEvCustomButtonClickMeth 28	Method event handler for OnCustomButtonClick event.
TELEvCustomButtonClickProc 628	Procedural event handler for OnCustomButtonClick event.
TELEvCustomDataRequestMeth 26	Method event handler for OnCustomDataRequest event.
TELEvCustomDataRequestProc 631	Procedural event handler for OnCustomDataRequest event.
TELEvCustomFileNameMeth 24	Method event handler for OnCustomFileName event.
TELEvCustomFileNameProc 633	Procedural event handler for OnCustomFileName event.
TELEvCustomWebFieldsRequestMeth 27	Method event handler for OnCustomWebFieldsRequest event.
TELEvCustomWebFieldsRequestProc 635	Procedural event handler for OnCustomWebFieldsRequest event.
TELEvExceptionActionMeth 16	Method event handler for OnExceptionAction event.
TELEvExceptionActionProc 638	Procedural event handler for OnExceptionAction event.
TELEvExceptionErrorMeth 18	Method event handler for OnExceptionError event.
TELEvExceptionErrorProc 642	Procedural event handler for OnExceptionError event.
TELEvExceptionNotifyMeth 14	Method event handler for OnExceptionNotify event.
TELEvExceptionNotifyProc 645	Procedural event handler for OnExceptionNotify event.
TELEvHelpButtonClickMeth 29	Method event handler for OnHelpButtonClick event.
TELEvHelpButtonClickProc 648	Procedural event handler for OnHelpButtonClick event.
TELEvPasswordRequestMeth 19	Method event handler for

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	OnPasswordRequest event.
TELEvPasswordRequestProc 651	Procedural event handler for OnPasswordRequest event.
TELEvRaiseMeth 12	Method event handler for OnRaise event.
TELEvRaiseProc 653	Procedural event handler for OnRaise event.
TELEvReportGenMeth 655	Method event handler for OnReportBeginGen/OnReportEndGen events.
TELEvReportGenMeth 30	Method event handler for OnReportBeginGen/ OnReportEndGen events.
TELEvReportGenProc 657	Procedural event handler for OnReportBeginGen/ OnReportEndGen event.
TELEvReportGenProc 657	Procedural event handler for OnReportBeginGen/OnReportEndGen event.
TELEvReproduceRequestMeth 23	Method event handler for OnReproduceRequest event.
TELEvReproduceRequestProc 659	Procedural event handler for OnReproduceRequest event.
TELEvZippedFilesRequestMeth 22	Method event handler for OnZippedFilesRequest event.
TELEvZippedFilesRequestProc 661	Procedural event handler for OnZippedFilesRequest event.
TELLocationInfo 662	A code location information.
TELUniversalSenderClass 669	Pointer to send method class.
TELUseParent 670	Used to indicate which location information should be used for include files.
TEncryptBuffer 671	Contains an arbitrary array of bytes.
TEurekaActionType 672	Action kind for exception processing.
TEurekaCallStack 678	Alias for TEurekaStackListV7.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EurekaLog CATCH EVERY BUG - EVERY TIME

TEurekaCallStackClass 678	Class of TEurekaBaseStackList.
TEurekaDebugDetail 678	Level of detalization.
TEurekaDebugDetails 680	Set of detalization levels.
TEurekaLogFileType 680	Type of file name.
TEurekaModuleType 683	Alias for TModuleType 709.
TEurekaStackList 684	Alias for TEurekaStackListV7.
TExceptionNotifyMode 684	Specifies work mode of exception manager.
TFreezeThreadClass 686	Class of freeze detection thread.
TInstruction 687	A CPU instruction.
TIssueIDType 690	Type of the issue ID in the bug tracker.
TLogMsgContent 691	Kind of data (content).
TLogMsgType 691	Kind of log entry.
TLogType 692	Common "type".
TMD5Hash 705	MD5 hash.
TMemoryError 705	Type of detected memory error.
TMemoryErrors 708	Defines set of possible memory errors.
TModuleType 709	Indicates kind of executable module.
TPointerRecord 710	Header for memory blocks allocated by EurekaLog.
TResponse 713	No description available
TRSAExport 717	Defines how to export/import RSA key.
TRSAImport 718	Alias to `TRSAExport`.
TRSAKey 719	Key for RSA encryption.
TRSASignature 720	Digital signature.
TSalt 721	Salt value for deriving key from password.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TScreeenshotHandle 721	Screenshot descriptor.
TSDBMHash 722	SDBM hash.
TSendResult 722	Result kind for exception processing.
TSendScreenshot 727	Specifies screen area.
TSHA1Hash 730	SHA-1 hash.
TSHA256Hash 730	SHA-2/SHA-256 hash.
TSignatureInfo 731	Information about digital signature of executable file.
TStreamConstructor 736	Function to Create a Compression/ Decompression stream
TSysErrorCode 737	A type for GetLastError/SetLastError.
TTEAKey 737	Key for TEA encryption.
TThreadKind 738	Indicates thread kind.
TTwofishInitVector 739	Initialization Vector (IV) for Twofish.
TTwofishKey 740	(Precalculated) Key for Twofish encryption.
TTwofishRAWKey 740	(RAW) Key for Twofish encryption.
TZipCompression 741	Zip compression method.
TZipEndOfCentralHeader 747	Final block written to zip file
TZipHeader 747	TZipHeader contains information about a file in a zip archive.

11.1 PCustomName

Defines buffer for storing name of memory block.

Unit

ETypes 138

Syntax

Code (Delphi)

```
PCustomName = ^TCustomName 622;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks
This is a pointer to TCustomName type.

See also
- TCustomName Type
- MemLeaksName Function 483
- IsMemLeaksEnabled Function 461

11.2 PELLocationInfo

A pointer to code location information.

Unit
EBaseClasses 58

Syntax

Code (Delphi)
```
PELLocationInfo = ^TELLocationInfo 662;
```

Remarks
This is a reference to code location information record.

See also
- TELLocationInfo Record. 662

11.3 PInstruction

A pointer to TInstruction 687.

Unit
EDisAsm 78

Syntax

Code (Delphi)
```
PInstruction = ^TInstruction 687;
```

Remarks
This is a pointer to platform-specific TInstruction 687 record.

See also
- TInstruction Type 687

11.4 PPointerRecord

A pointer to TPointerRecord type.

Unit
ETypes 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
PPointerRecord = ^TPointerRecord[710];
```

Remarks

This is a pointer to TPointerRecord record.

Warning

This data type is subject to change. Consider it to be undocumented.

See also

- TPointerRecord Type
- DumpAllocationsToFile Function[358]
- TDumpCompareFunc Type[622]
- TDumpFilterFunc Type[623]

11.5 PSalt

Pointer to salt.

Unit

EEncrypt[80]

Syntax

```
Code (Delphi)
PSalt = ^TSalt[721];
```

Remarks

Use InitSalt[438] to obtain new random salt. Pass the obtained value to TEADeriveKey[565] function or TwofishDeriveKey[580] function.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

See also

- TEADeriveKey Function[565]
- TwofishDeriveKey Function[580]
- InitSalt Function[438]

11.6 PtrUInt

Unsigned integer type with same size as `Pointer`.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 138

Syntax

```
Code (Delphi)
PtrUInt = NativeUInt;
```

Remarks
`PtrUInt` is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular `Cardinal` or `Integer` types.

EurekaLog introduces this type, because NativeUInt has wrong size in old IDE versions.

11.7 PTwofishInitVector

Pointer to IV for Twofish.

Unit
EEncrypt 80

Syntax

```
Code (Delphi)
PTwofishInitVector = ^TTwofishInitVector 739;
```

Remarks
Use TwofishInitIV 585 to obtain new random initialization vector. Pass the obtained value to TwofishEncrypt 583 and store it with encrypted data - to be passed into TwofishDecrypt 578 later.

```
Note
Initialization vector (IV) is a fixed-size input to a cryptographic primitive that is typically required to be random or pseudorandom. Randomization is crucial for encryption schemes to achieve semantic security, a property whereby repeated usage of the scheme under the same key does not allow an attacker to infer relationships between segments of the encrypted message.
```

See also
- TwofishEncrypt Function 583
- TwofishDecrypt Function 578
- TwofishInitIV Function 585

11.8 TAppType

Defines application type.

Unit
ETypes 138

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
Code (Delphi)
TAppType = String;
```

Remarks

This can be arbitrary string.

`ETypes` unit contains some predefined types - see `ETypes.at`*XYZ* constants.

See also

- IsAppType Function 442
- SetAppType Function 547

11.9 TBaseDialogClass

Pointer to dialog class.

Unit

EDialog 72

Syntax

```
Code (Delphi)
TBaseDialogClass = class of TBaseDialog;
```

Remarks

This type is used to register dialog classes into global list.

See also

- RegisterDialogClass 502
- RegisterDialogClassFirst 503
- FindDialogClass 399

11.10 TBlockType

Defines type of memory block.

Unit

ETypes 138

Syntax

```
Code (Delphi)
TBlockType = (
    btData 612,
    btObject 613,
    btCppObject 612,
    btString 613,
    btDynArray 612
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

btCppObject 612	The block is an instance of C++ object class.
btData 612	RAW memory block. This value is used when EurekaLog is not able to suggest more specific block type. Pointers, dynamically allocated records, buffers - are examples of such data.
btDynArray 612	The block is a dynamic array.
btObject 613	The block is an instance of TObject or descendant class.
btString 613	The block is a string. It can be ANSI or unicode string.

Remarks

This type is used to define type of memory block in several event handlers.

See also

- TMemLeaksError Type
- TMemLeaksAdd Type
- TMemLeaksLogBlockProc Type

11.10.1 btCppObject

The block is an instance of C++ object class.

Unit

ETypes 138

See also

- TBlockType Class 611

11.10.2 btData

RAW memory block. This value is used when EurekaLog is not able to suggest more specific block type. Pointers, dynamically allocated records, buffers - are examples of such data.

Unit

ETypes 138

See also

- TBlockType Class 611

11.10.3 btDynArray

The block is a dynamic array.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 138

See also
- TBlockType Class 611

11.10.4 btObject

The block is an instance of TObject or descendant class.

Unit
ETypes 138

See also
- TBlockType Class 611

11.10.5 btString

The block is a string. It can be ANSI or unicode string.

Unit
ETypes 138

See also
- TBlockType Class 611

11.11 TBugID

BugID type.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TBugID = LongWord;
```

Remarks
Current version of EurekaLog uses 4 bytes integer as BugID. The high 2 bytes are used as CRC value of string which identifies exception (see more). Lower 2 bytes are not used by EurekaLog and they are left for your own usage.

See also
- BugID concept

11.12 TCertificateInfo

Certificate information.

Unit
ESignature 131

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
TCertificateInfo = record
    Version: DWORD;
    Serial: String;
    Hash: String;
    Encryption: String;
    Subject: String;
    SubjectID: String;
    Issuer: String;
    IssuerID: String;
    NotBefore: TDateTime;
    NotAfter: TDateTime;
end;
```

Remarks

This record contains information about digital certificate.

See also

- TSignatureInfo Record
- SignatureGetInfo Function

11.12.1 Encryption

Encryption alg identifier.

Unit

ESignature

Syntax

```
Encryption: String;
```

Value

OID (object identifier) of the used algorythm.

Remarks

This field contains ID of the used encryption/signing algoryth.

Examples

`'1.2.840.113549.1.1.1'` RSA (szOID_RSA_RSA)

See also

- TCertificateInfo Record

11.12.2 Hash

Hash alg identifier.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ESignature 131

Syntax

Code (Delphi)

```
Hash: String;
```

Value
OID (object identifier) of the used algorythm.

Remarks
This field contains ID of the used hashing algoryth.

Examples
`'1.3.14.3.2.26'` SHA-1 (szOID_OIWSEC_sha1) `'2.16.840.1.101.3.4.2.1'` SHA-256 (szOID_NIST_sha256)

See also
- TCertificateInfo Record 613

11.12.3 Issuer

Issuer of certificate.

Unit
ESignature 131

Syntax

Code (Delphi)

```
Issuer: String;
```

Value
Human-readable issuer name.

Remarks
This field contains name of certificate authority which created (issued) the certificate.

Examples
```
'COMODO Code Signing CA 2'
```

See also
- IssuerID Field 615
- Subject Field 618
- TCertificateInfo Record 613

11.12.4 IssuerID

Unique issuer ID.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ESignature 131

Syntax

Code (Delphi)
```
IssuerID: String;
```

Value
Human-readable hex representation of issuer ID.

Remarks
This field contains unique ID for issuer. ID is arbitrary data which can identefy issuer. Hash is often used as ID.

This field is often not specified.

Examples
```
'B890D6C4C6677AB27D991E8B66522CCDE0872F11'
```

See also
- Issuer Field 615
- SubjectID Field 618
- TCertificateInfo Record 613

11.12.5 NotAfter

Ending date for certificate.

Unit
ESignature 131

Syntax

Code (Delphi)
```
NotAfter: TDateTime;
```

Value
Date and time after which the certificate is not valid.

Remarks
This field contains date and time after which the certificate is not valid. The time part is usually 23:59:59, which means ends of the day. E.g. start of the next day is not valid.

This field is only precise to seconds.

Examples
```
2017.11.30 23:59:59
```

See also
- TCertificateInfo Record 613

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.12.6 NotBefore

Starting date for certificate.

Unit
ESignature 131

Syntax

Code (Delphi)

```
NotBefore: TDateTime;
```

Value
Date and time before which the certificate is not valid.

Remarks
This field contains date and time before which the certificate is not valid. The time part is usually zero, which means start of the day.

This field is only precise to seconds.

Examples
```
2012.11.30 00:00:00
```

See also
- TCertificateInfo Record 613

11.12.7 Serial

Serial number of certificate.

Unit
ESignature 131

Syntax

Code (Delphi)

```
Serial: String;
```

Value
Human-readable hex-representation of serial number.

Remarks
This field contains serial number of the certificate that contains the public key.

The serial unique identifies the certificate.

Examples
```
'1998B8FD6B038A439C32F32CAF35C69B'
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TCertificateInfo Record 613

11.12.8 Subject

Subject name.

Unit

ESignature 131

Syntax

Code (Delphi)

```
Subject: String;
```

Value

Human-readable name of the subject.

Remarks

This field contains name of the certificate.

Examples

```
'EurekaLab s.a.s. di Fabio Dell'Aria and C.'
```

See also

- SubjectID Field 618
- Issuer Field 615
- TCertificateInfo Record 613

11.12.9 SubjectID

Unique subject ID.

Unit

ESignature 131

Syntax

Code (Delphi)

```
SubjectID: String;
```

Value

Human-readable hex represenation of subject ID.

Remarks

This field contains unique ID for subject. ID is arbitrary data which can identefy subject. Hash is often used as ID.

This field is often not specified.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples
'3DDA323E552DB1FDE5F4FBEE75D6D5B2B187EEDC'

See also
- Subject Field 618
- IssuerID Field 615
- TCertificateInfo Record 613

11.12.1(Version

Version of certificate.

Unit
ESignature 131

Syntax

Code (Delphi)
```
Version: DWORD;
```

Value
Integer version.

Remarks
Currently this can be 1 for CERT_V1, 2 for CERT_V2 or 3 for CERT_V3. 3 is most common.

Examples
3

See also
- TCertificateInfo Record 613

11.13 TCheckType

Function check kind.

Unit
EResLeaks 116

Syntax

Code (Delphi)
```
TCheckType = (
    ctNone 621,
    ctEqual 620,
    ctNotEqual 621,
    ctHResult 620,
    ctNTStatus 621,
    ctGetLastError 620
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

ctEqual 620	Function fails if return value equals to `CheckValue`
ctGetLastError 620	Function fails if return value equals to `CheckValue` **and** `GetLastError` returns value other than `ERROR_SUCCESS`
ctHResult 620	Function returns standard `HResult` value
ctNone 621	Function always successful
ctNotEqual 621	Function fails if return value wasn't equal to `CheckValue`
ctNTStatus 621	Function returns standard `NTSTATUS` value

Remarks

This type specifies how to determinate if function call was successful or not. This type is used by HookWin32API 434 function.

11.13.1 ctEqual

Function fails if return value equals to `CheckValue`

Unit

EResLeaks 116

See also

- TCheckType Class 619

11.13.2 ctGetLastError

Function fails if return value equals to `CheckValue` **and** `GetLastError` returns value other than `ERROR_SUCCESS`

Unit

EResLeaks 116

See also

- TCheckType Class 619

11.13.3 ctHResult

Function returns standard `HResult` value

Unit

EResLeaks 116

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also
- TCheckType Class 619

11.13.4 ctNone

Function always successful

Unit
EResLeaks 116

See also
- TCheckType Class 619

11.13.5 ctNotEqual

Function fails if return value wasn't equal to `CheckValue`

Unit
EResLeaks 116

See also
- TCheckType Class 619

11.13.6 ctNTStatus

Function returns standard `NTSTATUS` value

Unit
EResLeaks 116

See also
- TCheckType Class 619

11.14 TCRC16Hash

CRC-16 checksum.

Unit
EHash 97

Syntax

Code (Delphi)

```
TCRC16Hash = type Word;
```

Remarks
Data type to hold CRC-16 checksum. This is 2 byte (16 bit) value (Word).

See also
- CRC16Hash Function 345

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.15 TCRC32Hash

CRC-32 checksum.

Unit
EHash 97

Syntax

Code (Delphi)
```
TCRC32Hash = type Cardinal;
```

Remarks
Data type to hold CRC-32 checksum. This is 2 byte (16 bit) value (Word).

See also
- CRC32Hash Function 346

11.16 TCustomName

Defines buffer for storing name of memory block.

Unit
ETypes 138

Syntax

Code (Delphi)
```
TCustomName = array[0..59]of AnsiChar;
```

Remarks
This data type is used internally in EurekaLog to store name of each memory block. This type is used to define buffer for storing name inside memory block header. It is used only when memory leaks checking are enabled. Otherwise header will not contain name field.

You can name any memory block via MemLeaksName 483 function.

You can change size of this buffer if you have Enterprise version of EurekaLog. Alter the type's description and recompile EurekaLog.

See also
- MemLeaksName Function 483
- IsMemLeaksEnabled Function 461

11.17 TDumpCompareFunc

Comparator for memory blocks.

Unit
EMemLeaks 109

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TDumpCompareFunc = function(
   const AL, AR: PPointerRecord 608
): Integer;
```

Parameters

AL [in]
A pointer to internal undocumented PPointerRecord.

AR [in]
A pointer to internal undocumented PPointerRecord.

Return value

Return:

- 0: memory blocks are equal;
- > 0: AR is greater than AL;
- < 0: AL is greater than AR.

Remarks

This type is used to define sort order for DumpAllocationsToFile 358 function.

See also

- DumpAllocationsToFile Function 358

11.18 TDumpFilterFunc

Filter for memory blocks.

Unit

EMemLeaks 109

Syntax

```
Code (Delphi)
TDumpFilterFunc = function(
   const ABlock: PPointerRecord 608
): Boolean;
```

Parameters

ABlock [in]
A pointer to internal undocumented PPointerRecord.

Return value

`True` - if `ABlock` should be included in the report, `False` - otherwise.

Remarks

This type is used to filter memory blocks for DumpAllocationsToFile 358 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- DumpAllocationsToFile Function 358

11.19 TELEvAttachedFilesRequestMeth

Method event handler for OnAttachedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ATempFolder [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

AAttachedFiles [out]
List of attached file names. Use full absolute paths.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as separate files.

To include files into bug report file - use OnZippedFilesRequest event.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 504
- Procedural OnAttachedFilesRequest event handler 625
- OnZippedFilesRequest event handler 22
- EurekaLog component 33
- General events concept 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Other events 11

11.20 TELEvAttachedFilesRequestProc

Procedural event handler for OnAttachedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

ATempFolder [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

AAttachedFiles [out]
List of attached file names. Use full absolute paths.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as separate files.

To include files into bug report file - use OnZippedFilesRequest event.

See also

- Event handler registration 504
- Method OnAttachedFilesRequest event handler 21
- OnZippedFilesRequest event handler 661
- EurekaLog component 33
- General events concept 83
- Other events 11

11.21 TELEvCustomBugIDMeth

Method event handler for OnCustomBugID event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomBugIDMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ABugID, ACustomBugID: Word;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ABugID [in, out]
Primary BugID generated by EurekaLog.

ACustomBugID [out]
Custom BugID.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event handler to override BugID creation.

You can define your own sub-ID by altering `ACustomBugID`. Alternatively you may also replace BugID generated by EurekaLog.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 506
- Procedural OnCustomBugID event handler 626
- EurekaLog component 33
- General events concept 83
- Other events 11

11.22 TELEvCustomBugIDProc

Procedural event handler for OnCustomBugID event.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
TELEvCustomBugIDProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ABugID, ACustomBugID: Word;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

ABugID [in, out]
Primary BugID generated by EurekaLog.

ACustomBugID [out]
Custom BugID.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event handler to override BugID creation.

You can define your own sub-ID by altering `ACustomBugID`. Alternatively you may also replace BugID generated by EurekaLog.

See also

- Event handler registration 506
- Method OnCustomBugID event handler 25
- EurekaLog component 33
- General events concept 83
- Other events 11

11.23 TELEvCustomButtonClickMeth

Method event handler for OnCustomButtonClick event.

Unit

EEvents 83

Syntax

```
TELEvCustomButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACloseDialog: Boolean;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AOptions [in]
Current EurekaLog options.

ADialog [in]
Current dialog class (child class of `TBaseDialog`).

ACloseDialog [out]
True - close dialog after return, False (default) - do not close dialog.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked when end-user has clicked on custom button in error dialog. You should put your custom action code there. For example, if you setup custom button to be "check for updates" - then you should put your update checking code there.

Note

There is a similar event 29.
- Custom button is shown only when you specify it in options. It's shown for errors (i.e. bug reports).
- Help button is shown automatically whenever there is help context for known exception (which do not generate bug report).

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 507
- Procedural OnCustomButtonClick event handler 628
- OnHelpButtonClick event handler 29
- EurekaLog component 33
- General events concept 83
- Other events 11

11.24 TELEvCustomButtonClickProc

Procedural event handler for OnCustomButtonClick event.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TELEvCustomButtonClickProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACloseDialog: Boolean;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

AOptions [in]
Current EurekaLog options.

ADialog [in]
Current dialog class.

ACloseDialog [out]
True - close dialog after return, False (default) - do not close dialog.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked when end-user has clicked on custom button in error dialog. You should put your custom action code there. For example, if you setup custom button to be "check for updates" - then you should put your update checking code there.

Note

There is a similar event 648.

- Custom button is shown only when you specify it in options. It's shown for errors (i.e. bug reports).
- Help button is shown automatically whenever there is help context for known exception (which do not generate bug report).

See also

- Event handler registration 507
- Method OnCustomButtonClick event handler 28
- OnHelpButtonClick event handler 648
- EurekaLog component 33
- General events concept 83
- Other events 11

11.25 TELEvCustomDataRequestMeth

Method event handler for OnCustomDataRequest event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomDataRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   ADataFields: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ALogBuilder [in]
Log builder class, representing bug report.

ADataFields [out]
Custom fields to add into bug report (as `name=value` pairs).

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before saving bug report to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `ADataFields.Values['Field-Name'] := 'Field-Value'`. Field name can be arbitrary and are choosen by you.

Warning

Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 508
- Procedural OnCustomDataRequest event handler 631
- EurekaLog component 33
- General events concept 83
- Other events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.26 TELEvCustomDataRequestProc

Procedural event handler for OnCustomDataRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomDataRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   ADataFields: TStrings;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

ALogBuilder [in]
Log builder class, representing bug report.

ADataFields [out]
Custom fields to add into bug report (as `name=value` pairs).

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before saving bug report to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `ADataFields.Values['Field-Name'] := 'Field-Value'`. Field name can be arbitrary and are choosen by you.

Warning

Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration 508
- Method OnCustomDataRequest event handler 26
- EurekaLog component 33
- General events concept 83
- Other events 11

11.27 TELEvCustomFileNameMeth

Method event handler for OnCustomFileName event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomFileNameMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AFileType: TEurekaLogFileType 680;
   var AFileName: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AFileType [in]
Kind of file name to modify.

AFileName [in, out]
Actual file name that may be changed. Use full absolute file path.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify file names.

Even though you can relocate file in arbitrary folder - it's highly recommended to keep original folder and extension.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration 509
- Procedural OnCustomFileName event handler 633
- EurekaLog component 33
- General events concept 83
- Other events 11

11.28 TELEvCustomFileNameProc

Procedural event handler for OnCustomFileName event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomFileNameProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AFileType: TEurekaLogFileType 680;
   var AFileName: String;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

AFileType [in]
Kind of file name to modify.

AFileName [in, out]
Actual file name that may be changed. Use full absolute file path.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify file names.

Even though you can relocate file in arbitrary folder - it's highly recommended to keep original folder and extension.

See also

- Event handler registration 509
- Method OnCustomFileName event handler 24
- EurekaLog component 33
- General events concept 83
- Other events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.29 TELEvCustomWebFieldsRequestMeth

Method event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomWebFieldsRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ASender: TObject;
   AWebFields: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

ASender [in]
Current sender class (child class of `TELUniversalSender`).

AWebFields [in]
Web fields to send to web-site (as `name=value` pairs).

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before sending to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `AWebFields.Values['Field-Name'] := 'Field-Value'`. Field name depends on used send method.

For example, if you are sending to Mantis - then field name would be custom field name as it appears in Mantis configuration (e.g. like `'cf_count'`).

This event is not called if send method does not need / support custom fields.

Note

`AExceptionInfo` param will be missing if sending is performed outside of exception processing. E.g. if sending is called manually.

Warning

Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 510
- Procedural OnCustomWebFieldsRequest event handler 635
- EurekaLog component 33
- General events concept 83
- Other events 11

11.30 TELEvCustomWebFieldsRequestProc

Procedural event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvCustomWebFieldsRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ASender: TObject;
   AWebFields: TStrings;
   var ACallNextHandler: Boolean
);
```

Parameters

`ACustom` [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

`AExceptionInfo` [in, optional]
Information about exception. Can be `nil`.

`ASender` [in]
Current sender class (child class of `TELUniversalSender`).

`AWebFields` [in]
Web fields to send to web-site (as `name=value` pairs).

`ACallNextHandler` [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked before sending to allow you to supply custom fields. Fields should be supplied as `name=value` pairs. We recommend to use the following construct: `AWebFields.Values['Field-Name'] := 'Field-Value'`. Field name depends on used send method.

For example, if you are sending to Mantis - then field name would be custom field name as it appears in

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Mantis configuration (e.g. like `'cf_count'`).

This event is not called if send method does not need / support custom fields.

Note
`AExceptionInfo` param will be missing if sending is performed outside of exception processing. E.g. if sending is called manually.

Warning
Do not confuse custom fields (`OnCustomDataRequest` event) and web fields (`OnCustomWebFieldsRequest` event). Custom fields are inserted into bug report file. Web fields are not stored in bug report, web fields are POST-ed to bug tracker / web-site via API or HTTP request.

Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 510
- Method OnCustomWebFieldsRequest event handler 27
- EurekaLog component 33
- General events concept 83
- Other events 11

11.31 TELEvExceptionActionMeth

Method event handler for OnExceptionAction event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionActionMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AEurekaAction: TEurekaActionType 672;
   const AAdditionalInfo: String;
   var AExecute: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

`AExceptionInfo` [in, optional]
Information about exception. Can be `nil`.

`AEurekaAction` [in]

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Action to perform.

AAdditionalInfo [in, optional]
Additional information about action (see remarks).

AExecute [out]
"*Before-like*" actions: `True` - (default) perform action, `False` - cancel/skip action;

"*After-like*" actions: `True` - action is successful, `False` - action was unsuccessful;

Note
This argument is ignored for atSendCancelled 676.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Tip
Set to `False`, if you set `AExecute` to `False`.

Remarks

OnExceptionAction event is fired for any action with exception during exception processing: saving to bug report, sending via Internet, etc. You can use this event to alter actions, cancel/skip them, etc.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 677 and atSent 677 actions. For these actions - `AAdditionalInfo` argument holds send engine class name. This information is useful if you use several send methods (for example, HTTP and e-mail). See also 41.

For "before"-like actions: `AExecute` will determinate if EurekaLog should execute the action. You can use this to skip certain actions (e.g. skip sending, skip dialog, skip saving bug report, etc.).

For "after"-like actions: `AExecute` will determinate if EurekaLog should consider action as succeeded (or failed). You can use this to skip errors.

Note
Basically, for most actions EurekaLog follow such pattern (pseudo-code):

Code (Delphi)

```
repeat
          MakeCall := DoEventExceptionAction(... "before" ...);
          try
          if MakeCall then
          Result := PerformAction
          else
          Result := Cancelled;
          except
          Result := UnknownError;
          end;
          Result := DoEventExceptionAction(... "after" ...);

          if Failed(Result) then
          Retry := DoEventExceptionError(...)
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
          else
          Retry := False;
        until not Retry;
```

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event handler will be c
except
   Application.HandleException();
       // - OnExceptionNotify will be called
      // - set of OnExceptionAction events will be called
       // (one event for each action (saving, sending, etc).)
   end;
```

See also

- Event handler registration 512
- Procedural OnExceptionAction event handler 638
- EurekaLog component 33
- General events concept 83
- Other events 11

11.32 TELEvExceptionActionProc

Procedural event handler for OnExceptionAction event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionActionProc = procedure(
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    const ACustom: Pointer;
    AExceptionInfo: TEurekaExceptionInfo 180;
    const AEurekaAction: TEurekaActionType 672;
    const AAdditionalInfo: String;
    var AExecute: Boolean;
    var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

AEurekaAction [in]
Action to perform.

AAdditionalInfo [in, optional]
Additional information about action (see remarks).

AExecute [out]
"*Before-like*" actions: `True` - (default) perform action, `False` - cancel/skip action;

"*After-like*" actions: `True` - action is successful, `False` - action was unsuccessful;

Note
This argument is ignored for atSendCancelled 676.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Tip
Set to `False`, if you set `AExecute` to `False`.

Remarks

OnExceptionAction event is fired for any action with exception during exception processing: saving to bug report, sending via Internet, etc. You can use this event to alter actions, cancel/skip them, etc.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 677 and atSent 677 actions. For these actions - `AAdditionalInfo` argument holds send engine class name. This information is useful if you use several send methods (for example, HTTP and e-mail). See also 41.

For "before"-like actions: `AExecute` will determinate if EurekaLog should execute the action. You can use this to skip certain actions (e.g. skip sending, skip dialog, skip saving bug report, etc.).

For "after"-like actions: `AExecute` will determinate if EurekaLog should consider action as succeeded (or failed). You can use this to skip errors.

Note
Basically, for most actions EurekaLog follow such pattern (pseudo-code):

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
repeat
          MakeCall := DoEventExceptionAction(... "before" ...);
          try
          if MakeCall then
          Result := PerformAction
          else
          Result := Cancelled;
          except
          Result := UnknownError;
          end;
          Result := DoEventExceptionAction(... "after" ...);

          if Failed(Result) then
          Retry := DoEventExceptionError(...)
          else
          Retry := False;
         until not Retry;
```

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event handler will be c
except
   Application.HandleException();
       // - OnExceptionNotify will be called
      // - set of OnExceptionAction events will be called
       // (one event for each action (saving, sending, etc).)
   end;
```

See also

- Event handler registration 512
- Method OnExceptionAction event handler 16
- EurekaLog component 33
- General events concept 83
- Other events 11

11.33 TELEvExceptionErrorMeth

Method event handler for OnExceptionError event.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TELEvExceptionErrorMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AEurekaAction: TEurekaActionType 672;
   const AAdditionalInfo: String;
   var ARetry: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

AEurekaAction [in]
Failed action.

AAdditionalInfo [in]
Reason why action has failed.

ARetry [out]
`True` - repeat failed action, `False` - (default) fail operation.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Tip

Set to `False`, if you set `ARetry` to `True`.

Remarks

OnExceptionError event is fired for any failed operation during exception processing: saving to bug report, sending via Internet, etc. You can use this event to correct settings and retry the operation or to use a fallback fail code, etc.

`AExceptionInfo` represents original exception which is being processed. It may be `nil`.

`AAdditionalInfo` represents secondary error which failed `AEurekaAction` step.

Tip

Be sure to do something before switching `ARetry` to `True`. Repeating failed action without any attempt to correct settings / environment will simply fail again.

Warning

Be careful when setting `ARetry` to `True`: the failed action will be repeated and may raise the same error again. Ensure you have some sort of check to avoid infinite loop.

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 513
- Procedural OnExceptionError event handler 642
- EurekaLog component 33
- General events concept 83
- Other events 11

11.34 TELEvExceptionErrorProc

Procedural event handler for OnExceptionError event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionErrorProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   const AEurekaAction: TEurekaActionType 672;
   const AAdditionalInfo: String;
   var ARetry: Boolean;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

AEurekaAction [in]
Failed action.

AAdditionalInfo [in]
Reason why action has failed.

ARetry [out]
`True` - repeat failed action, `False` - (default) fail operation.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Tip
Set to `False`, if you set `ARetry` to `True`.

Remarks

OnExceptionError event is fired for any failed operation during exception processing: saving to bug report, sending via Internet, etc. You can use this event to correct settings and retry the operation or to use a fallback fail code, etc.

`AExceptionInfo` represents original exception which is being processed. It may be `nil`.

`AAdditionalInfo` represents secondary error which failed `AEurekaAction` step.

Tip
Be sure to do something before switching `ARetry` to `True`. Repeating failed action without any attempt to correct settings / environment will simply fail again.

Warning
Be careful when setting `ARetry` to `True`: the failed action will be repeated and may raise the same error again. Ensure you have some sort of check to avoid infinite loop.

Caution!
Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 513
- Method OnExceptionError event handler 18
- EurekaLog component 33
- General events concept 83
- Other events 11

11.35 TELEvExceptionNotifyMeth

Method event handler for OnExceptionNotify event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvExceptionNotifyMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AHandle: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AHandle [out]
`True` - (default) process exception with EurekaLog, `False` - process exception with standard exception handler

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

> Tip
>
> Set `ACallNextToHandler` to `False`, if you set `AHandle` to `False`.

Remarks

By default: OnExceptionNotify event is fired for any exception which wasn't handled by the code.

> Note
>
> If "Catch handled exception" option is enabled - this event will also fire for any handled exception.
>
> You can determinate the kind of exception (handled or unhandled) by analyzing properties of `AExceptionInfo` argument (see Handled property).

You can use this event to disable processing of certain exceptions by EurekaLog.

> Tip
>
> You can also use exception filters instead of this event.

Use OnExceptionAction 638 event to fine-control processing of exceptions.

> **Caution!**
>
> Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
finally
    ...
end;
   // - OnExceptionNotify will be called
```

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
except
    ...
end;
   // - OnExceptionNotify will not be called, since exception was handle
```

See this article 83 for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure MyExceptionNotify(const ACustom: Pointer; AExceptionInfo: TEur
begin
    // Disable exception processing by EurekaLog
    Handle := False;
    ACallNextHandler := False;
end;
```

See also

- Event handler registration 514
- Procedural OnExceptionNotify event handler 645
- EurekaLog component 33
- General events concept 83
- Other events 11

11.36 TELEvExceptionNotifyProc

Procedural event handler for OnExceptionNotify event.

Unit

EEvents 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TELEvExceptionNotifyProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AHandle: Boolean;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

AHandle [out]
`True` - (default) process exception with EurekaLog, `False` - process exception with standard exception handler

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

> **Tip**
>
> Set `ACallNextToHandler` to `False`, if you set `AHandle` to `False`.

Remarks

By default: OnExceptionNotify event is fired for any exception which wasn't handled by the code.

> **Note**
>
> If "Catch handled exception" option is enabled - this event will also fire for any handled exception.
>
> You can determinate the kind of exception (handled or unhandled) by analyzing properties of `AExceptionInfo` argument (see Handled property).

You can use this event to disable processing of certain exceptions by EurekaLog.

> **Tip**
>
> You can also use exception filters instead of this event.

Use OnExceptionAction 638 event to fine-control processing of exceptions.

> **Caution!**
>
> Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
finally
    ...
end;
   // - OnExceptionNotify will be called
```

Code (Delphi)

```
try
   raise Exception.Create('Error'); // - OnRaise event
except
    ...
end;
   // - OnExceptionNotify will not be called, since exception was handle
```

See this article 83 for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure MyExceptionNotify(const ACustom: Pointer; AExceptionInfo: TEur
begin
    // Disable exception processing by EurekaLog
    Handle := False;
    ACallNextHandler := False;
end;
```

See also

- Event handler registration 514
- Method OnExceptionNotify event handler 14
- EurekaLog component 33
- General events concept 83
- Other events 11

11.37 TELEvHelpButtonClickMeth

Method event handler for OnHelpButtonClick event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvHelpButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACallNextHandler: Boolean
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AOptions [in]
Current EurekaLog options.

ADialog [in]
Dialog class.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked when end-user has clicked on "Help" button in error dialog.

Help button is displayed automatically for known exceptions (i.e. exceptions that are not errors; they do not generate bug reports).

Note

There is a similar event 28.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 515
- Procedural OnHelpButtonClick event handler 648
- OnCustomButtonClick event handler 28
- EurekaLog component 33
- General events concept 83
- Other events 11
- RaiseExpected routine 499

11.38 TELEvHelpButtonClickProc

Procedural event handler for OnHelpButtonClick event.

Unit

EEvents 83

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
TELEvHelpButtonClickProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ADialog: TObject;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

AOptions [in]
Current EurekaLog options.

ADialog [in]
Dialog class.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is invoked when end-user has clicked on "Help" button in error dialog.

Help button is displayed automatically for known exceptions (i.e. exceptions that are not errors; they do not generate bug reports).

Note
There is a similar event 628.

See also

- Event handler registration 515
- Method OnHelpButtonClick event handler 29
- OnCustomButtonClick event handler 628
- EurekaLog component 33
- General events concept 83
- Other events 11
- RaiseExpected routine 499

11.39 TELEvPasswordRequestMeth

Method event handler for OnPasswordRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvPasswordRequestMeth = procedure(
   const AModule: HMODULE;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
   const APID: PGUID;
   const APasswordHash: Pointer;
   var APassword: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AModule [in]
Module handle which password is retrieved.

APID [in, optional]
Pointer to unique project ID. Can be used to pick password from cache. This parameter is always assigned for EurekaLog 7+ modules.

APasswordHash [in, optional]
Pointer to EHash.TSHA1Hash 730. SHA1 hash of the UTF8-encoded password. Can be used to validate password. This parameter is always assigned for EurekaLog 7+ modules.

APassword [out]
Password for debug information.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

EurekaLog includes a OnPasswordRequest event for interaction with the user program when a raised exception is handled by EurekaLog and debug information was saved as encrypted. This event allow EurekaLog to obtain the exact password required to can decrypt all the encrypted data, otherwise all the data will be store in the bug report file in encrypted form, also error dialogs will show bug report in encrypted form.

The developer of application may use EurekaLog Viewer tool to decrypt encrypted bug reports.

You can use this event to supply your password to EurekaLog. By default this password is not stored in your executables. It's impossible to decrypt encrypted debug information without a password.

You can either ask user for password at run-time or retrieve it from application's configuration.

Note
This event will not be called if password was obtained by some other means (for example, loaded from local cache, or supplied via `--el_password=` cmd line argument, etc.).

Warning
Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 516
- Method OnPasswordRequest event handler 651
- EurekaLog component 33

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- General events concept 83
- Other events 11

11.40 TELEvPasswordRequestProc

Procedural event handler for OnPasswordRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvPasswordRequestProc = procedure(
   const ACustom: Pointer;
   const AModule: HMODULE;
   const APID: PGUID;
   const APasswordHash: Pointer;
   var APassword: String;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AModule [in]
Module handle which password is retrieved.

APID [in, optional]
Pointer to unique project ID. Can be used to pick password from cache. This parameter is always assigned for EurekaLog 7+ modules.

APasswordHash [in, optional]
Pointer to EHash.TSHA1Hash 730. SHA1 hash of the UTF8-encoded password. Can be used to validate password. This parameter is always assigned for EurekaLog 7+ modules.

APassword [out]
Password for debug information.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

EurekaLog includes a OnPasswordRequest event for interaction with the user program when a raised exception is handled by EurekaLog and debug information was saved as encrypted. This event allow EurekaLog to obtain the exact password required to can decrypt all the encrypted data, otherwise all the data will be store in the bug report file in encrypted form, also error dialogs will show bug report in encrypted form.

The developer of application may use EurekaLog Viewer tool to decrypt encrypted bug reports.

You can use this event to supply your password to EurekaLog. By default this password is not stored in your executables. It's impossible to decrypt encrypted debug information without a password.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

You can either ask user for password at run-time or retrieve it from application's configuration.

Note
This event will not be called if password was obtained by some other means (for example, loaded from local cache, or supplied via `--el_password=` cmd line argument, etc.).

See also

- Event handler registration 516
- Method OnPasswordRequest event handler 19
- EurekaLog component 33
- General events concept 83
- Other events 11

11.41 TELEvRaiseMeth

Method event handler for OnRaise event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvRaiseMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

`AExceptionInfo` [in, optional]
Information about exception. Can be `nil`.

`ACallNextHandler` [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

OnRaise event is fired for each raised exception. You can use this event to collect additional information about exception, to change existing information or place any other custom logic.

Important!
Do not place exception processing code into event handler for this event. Use OnExceptionNotify 14 event instead.

Note
OnRaise event is invoked when EurekaLog detects that a new exception is raised. Depending on project configuration and environment - it can happen before or after an exception is actually thrown by CPU. This means that not all exception information may be collected. For example, a call stack may be `nil` - because building call stack for the exception requires CPU context record - which is not available until exception is actually thrown by CPU.

Note

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

OnRaise event is wrapped into OnActionNotify events with atCollectingExceptionInfo/ atCollectedExceptionInfo.

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

Examples

Code (Delphi)

```
raise Exception.Create('Error'); // - event will be raised here
```

See this article 83 for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure TMyObject.OnRaise(AExceptionInfo: TEurekaExceptionInfo; var AC
begin
   // Collect additional information for exception
   AExceptionInfo.AdditionalInfo := 'Sample additional data';
end;
```

See also

- Event handler registration 517
- Procedural OnRaise event handler 653
- EurekaLog component 33
- General events concept 83
- Other events 11

11.42 TELEvRaiseProc

Procedural event handler for OnRaise event.

Unit

EEvents 83

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
TELEvRaiseProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in, optional]
Information about exception. Can be `nil`.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

OnRaise event is fired for each raised exception. You can use this event to collect additional information about exception, to change existing information or place any other custom logic.

Important!

Do not place exception processing code into event handler for this event. Use OnExceptionNotify 645 event instead.

Note

OnRaise event is invoked when EurekaLog detects that a new exception is raised. Depending on project configuration and environment - it can happen before or after an exception is actually thrown by CPU. This means that not all exception information may be collected. For example, a call stack may be `nil` - because building call stack for the exception requires CPU context record - which is not available until exception is actually thrown by CPU.

Note

OnRaise event is wrapped into OnActionNotify events with atCollectingExceptionInfo/ atCollectedExceptionInfo.

Caution!

Any event is called from the context of exception thread, which may be not the main thread of application. Thus, it's highly recommended to avoid using any global-like data and to avoid access VCL objects. If you still need to do any of that - please use proper thread synchronization for globals and Synchronize method for VCL access.

Examples

Code (Delphi)

```
raise Exception.Create('Error'); // - event will be raised here
```

See this article 83 for example of registering event handlers.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Sample event handler may look like this:

Code (Delphi)

```
procedure OnRaise(const ACustom: Pointer; AExceptionInfo: TEurekaExcepti
begin
   // Collect additional information for exception
   AExceptionInfo.AdditionalInfo := 'Sample additional data';
end;
```

See also

- Event handler registration 517
- Method OnRaise event handler 12
- EurekaLog component 33
- General events concept 83
- Other events 11

11.43 TELEvReportGenMeth

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TELEvReportGenMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TBaseLogBuilder 152;
   var CallNextHandler: Boolean
) of object;
```

Parameters

ALogBuilder [in]
Log builder class.

ACallNextHandler [out]
True (default) - call next registered event handler, False - do not call next event handler.

Remarks

This event is called before creating bug report. You can use this event to intercept creation of bug reports. The same event handler type is used for both OnReportBegtinGen and OnReportEndGen events.

See also

- Event handler registration
- Procedural OnReportBeginGen/OnReportEndGen event handler 657
- EurekaLog component 33
- General events concept 83
- Other events 11

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.44 TELEvReportGenMeth

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReportGenMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ALogBuilder [in]
Log builder class.

AOptions [in]
Current EurekaLog options.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is called before creating bug report. You can use this event to intercept creation of bug reports. The same event handler type is used for both OnReportBegtinGen and OnReportEndGen events.

`ALogBuilder` is TBaseLogBuilder 152 instance. You need to use type cast.

Note

A more convenient case of this type 655 is declared in ELogBuilder 101 unit. Use it instead of this type whenever possible. Type from ELogBuilder doesn't require you to use typecast.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration
- Procedural OnReportBeginGen/OnReportEndGen event handler 18
- EurekaLog component 33
- General events concept 83

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Other events 11

11.45 TELEvReportGenProc

Procedural event handler for OnReportBeginGen/OnReportEndGen event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReportGenProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TObject;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

ALogBuilder [in]
Log builder class.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

This event is called before creating bug report. You can use this event to intercept creation of bug reports. The same event handler type is used for both OnReportBegtinGen and OnReportEndGen events.

`ALogBuilder` is TBaseLogBuilder 152 instance. You need to use type cast.

Note

A more convenient case of this type 657 is declared in ELogBuilder 101 unit. Use it instead of this type whenever possible. Type from ELogBuilder doesn't require you to use typecast.

See also

- Event handler registration
- Method OnReportBeginGen/OnReportEndGen event handler 30
- EurekaLog component 33
- General events concept 83
- Other events 11

11.46 TELEvReportGenProc

Procedural event handler for OnReportBeginGen/OnReportEndGen event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
TELEvReportGenProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   ALogBuilder: TBaseLogBuilder 152;
   var CallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

ALogBuilder [in]
Log builder class.

ACallNextHandler [out]
True (default) - call next registered event handler, False - do not call next event handler.

Remarks

This event is called before creating bug report. You can use this event to intercept creation of bug reports. The same event handler type is used for both OnReportBegtinGen and OnReportEndGen events.

See also

- Event handler registration
- Method OnReportBeginGen/OnReportEndGen event handler 655
- EurekaLog component 33
- General events concept 83
- Other events 11

11.47 TELEvReproduceRequestMeth

Method event handler for OnReproduceRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReproduceRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AReproduceText: String;
   var ACallNextHandler: Boolean
) of object;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AExceptionInfo [in]
Information about exception.

AReproduceText [in, out]
Text to be used as "steps to reproduce".

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify "steps to reproduce". A typical example would be appending run-time information to steps to reproduce. Alternatively, you may use OnCustomData event handler.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 518
- Procedural OnReproduceRequest event handler 659
- EurekaLog component 33
- General events concept 83
- Other events 11

11.48 TELEvReproduceRequestProc

Procedural event handler for OnReproduceRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvReproduceRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   var AReproduceText: String;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

AReproduceText [in, out]
Text to be used as "steps to reproduce".

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

Use this event to assign/modify "steps to reproduce". A typical example would be appending run-time information to steps to reproduce. Alternatively, you may use OnCustomData event handler.

See also

- Event handler registration 518
- Method OnReproduceRequest event handler 23
- EurekaLog component 33
- General events concept 83
- Other events 11

11.49 TELEvZippedFilesRequestMeth

Method event handler for OnZippedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ATempFolder [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

AAttachedFiles [out]
List of attached file names. Use full absolute paths.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as inside packed report (.elp file).

To include files into bug report file - use OnAttachedFilesRequest event.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Warning

Event handlers registered via `TEurekaLogEvents` component will be active only when corresponding form is live (created). Such handlers will not be called before form is created or after it was destroyed. For this reason you may want to register your event handlers via code. For example, it would be a bad idea to use `TEurekaLogEvents` component for implementing your error dialog, because your dialog will not be displayed for exceptions outside form (e.g. initialization/finalization exceptions, as well as exceptions in form's constructor).

See also

- Event handler registration 519
- Procedural OnZippedFilesRequest event handler 661
- OnAttachedFilesRequest event handler 21
- EurekaLog component 33
- General events concept 83
- Other events 11

11.50 TELEvZippedFilesRequestProc

Procedural event handler for OnZippedFilesRequest event.

Unit

EEvents 83

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 180;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]
Custom user data parameter. It's the same pointer as it was passed to RegisterEvent-function.

AExceptionInfo [in]
Information about exception.

ATempFolder [in]
Temp folder name with trailing path delimiter. You can use it to store any temp files. All files in this folder will be deleted automatically.

AAttachedFiles [out]
List of attached file names. Use full absolute paths.

ACallNextHandler [out]
`True` (default) - call next registered event handler, `False` - do not call next event handler.

Remarks

You can use this event to send arbitrary files with bug report. Files will be send as inside packed report (.elp

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

file).

To include files into bug report file - use OnAttachedFilesRequest event.

See also

- Event handler registration 519
- Method OnZippedFilesRequest event handler 22
- OnAttachedFilesRequest event handler 625
- EurekaLog component 33
- General events concept 83
- Other events 11

11.51 TELLocationInfo

A code location information.

Unit

EBaseClasses 58

Syntax

Code (Delphi)

```
TELLocationInfo = record
    Address 663: Pointer;
    DebugDetail 664: TEurekaDebugDetail 678;
    Offset 666: DWORD;
    Module 665: HMODULE;
    ModuleName 666: String;
    ModuleType 666: TEurekaModuleType 683;
    UnitName 669: String;
    SourceName 669: String;
    ClassName 663: String;
    ProcedureName 668: String;
    LineNumber 665: Integer;
    OffsetFromProcName 667: Integer;
    ProcOffsetLine 668: Integer;
    OffsetFromLineNumber 667: Integer;
    ParentProcOffsetLine: Integer;
    ParentLineNumber: Integer;
    ParentOffsetFromLineNumber: Integer;
    ParentSourceName: String;
    DebugInfo 664: TELDebugInfoSource 164;
    BinaryFileName 663: String;
end;
```

Remarks

This record holds information about one specific code location (method, function, line, etc.).

It can be obtained by various routines from EDebugInfo unit 423.

See also

- GetLocationInfo Function 423

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.51.1 Address

RAW pointer.

Unit
EBaseClasses 58

Syntax

Code (Delphi)
```
Address: Pointer;
```

Value
Code address.

Remarks
Absolute code address. This is the original address to code which was used to obtain information.

See also
- TELLocationInfo Record 662

11.51.2 BinaryFileName

Debug information file source.

Unit
EBaseClasses 58

Syntax

Code (Delphi)
```
BinaryFileName: String;
```

Value
Full absolute file name.

Remarks
Name of the file from which debug information was loaded. Typically it is the same as ModuleName. It will be different if debug information was loaded from external file (such as .map, .jdbg, etc.).

See also
- TELLocationInfo Record 662

11.51.3 ClassName

Class name.

Unit
EBaseClasses 58

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
ClassName: String;
```

Value
Class symbol name.

Remarks
This is class name to which method with code belongs to. It will be empty when code is not inside method.

See also
- TELLocationInfo Record 662

11.51.4 DebugDetail

Detalization level.

Unit
EBaseClasses 58

Syntax

```
Code (Delphi)
DebugDetail: TEurekaDebugDetail 678;
```

Value
Level of detalization.

Remarks
Indicates level of debug info's detalization. Basically, it indicates what fields in this record are filled.

See also
- TELLocationInfo Record 662

11.51.5 DebugInfo

Debug information class.

Unit
EBaseClasses 58

Syntax

```
Code (Delphi)
DebugInfo: TELDebugInfoSource 164;
```

Value
Debug information class.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks
Specifies class which was used to load debug information.

See also
- TELLocationInfo Record 662

11.51.6 LineNumber

Line number.

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
LineNumber: Integer;
```

Value
Line number in source file.

Remarks
This is number of the text line to which code belongs to.

See also
- TELLocationInfo Record 662

11.51.7 Module

Module.

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
Module: HMODULE;
```

Value
Start of the module.

Remarks
This is the executable module (.exe, .dll, .bpl) to which code belongs to.

See also
- TELLocationInfo Record 662

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.51.8 ModuleName

Module name.

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
ModuleName: String;
```

Value
Full absolute file name.

Remarks
This is the name of the Module.

See also
- TELLocationInfo Record 662

11.51.9 ModuleType

Module type.

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
ModuleType: TEurekaModuleType 683;
```

Value
Kind of the Module.

Remarks
This indicates if Module is .exe, .dll, or .bpl.

See also
- TELLocationInfo Record 662

11.51.1(Offset

Module offset.

Unit
EBaseClasses 58

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
Offset: DWORD;
```

Value
= Address - Module.

Remarks
This is offset of the Address from the start of the Module in bytes.

See also
- TELLocationInfo Record 662

11.51.11 OffsetFromLineNumber

Offset from line.

Unit
EBaseClasses 58

Syntax

```
OffsetFromLineNumber: Integer;
```

Value
= Address - StartOfLine.

Remarks
Offset from start of the line in bytes.

See also
- TELLocationInfo Record 662

11.51.12 OffsetFromProcName

Offset from procedure.

Unit
EBaseClasses 58

Syntax

```
OffsetFromProcName: Integer;
```

Value
= Address - StartOfRoutine.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks
Offset from start of the routine in bytes.

See also
- TELLocationInfo Record 662

11.51.13 ProcedureName

Routine name.

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
ProcedureName: String;
```

Value
Method name when Class is not empty. Function or procedure name when Class is empty.

Remarks
This is the name of a routine (method/function/procedure) to code belongs to.

See also
- TELLocationInfo Record 662

11.51.14 ProcOffsetLine

Offset from procedure (lines).

Unit
EBaseClasses 58

Syntax

Code (Delphi)

```
ProcOffsetLine: Integer;
```

Value
= LineOf(Address) - LineOf(StartOfRoutine).

Remarks
Difference in line numbers between code (Address) and start of the routine.

See also
- TELLocationInfo Record 662

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.51.15 SourceName

Source name.

Unit
EBaseClasses 58

Syntax

```
Code (Delphi)
SourceName: String;
```

Value
Unit file name (not unit name!).

Remarks
This is file name (without path) of the unit. Such as `Unit1.pas`.

See also
- TELLocationInfo Record 662

11.51.16 UnitName

Unit name.

Unit
EBaseClasses 58

Syntax

```
Code (Delphi)
UnitName: String;
```

Value
Unit symbol name (not file name!).

Remarks
This is unit name to which code belongs to. Such as `Unit1`.

See also
- TELLocationInfo Record 662

11.52 TELUniversalSenderClass

Pointer to send method class.

Unit
ESend 120

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELUniversalSenderClass = class of TELUniversalSender;
```

Remarks

This type is used to register send method classes into global list.

See also

- RegisterSender 520
- RegisterSenderFirst 520
- FindSender 401
- EnumSenders 387

11.53 TELUseParent

Used to indicate which location information should be used for include files.

Unit

EDebugInfo 68

Syntax

```
Code (Delphi)
TELUseParent = (
    eupNone 671,
    eupChild 671,
    eupParent 671
);
```

Members

eupChild 671	Use debug information for the included file.
eupNone 671	Use debug information for the non-include file.
eupParent 671	Use debug information for the file with {$I} directive.

Remarks

This type is used to indicate which debug information should be used when converting debug information to string.

See also

- LocationToStr Function 478

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.53.1 eupChild

Use debug information for the included file.

Unit
EDebugInfo 68

See also
- TELUseParent Class 670

11.53.2 eupNone

Use debug information for the non-include file.

Unit
EDebugInfo 68

See also
- TELUseParent Class 670

11.53.3 eupParent

Use debug information for the file with {$I} directive.

Unit
EDebugInfo 68

See also
- TELUseParent Class 670

11.54 TEncryptBuffer

Contains an arbitrary array of bytes.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
TEncryptBuffer = EWinCrypt.DATA_BLOB;
```

Remarks
This is simply a combined Pointer/Size record.

pbData field points to the data buffer.

cbData field contains the size of data in pbData in bytes.

Note

Functions from EEncrypt 80 unit will allocate `TEncryptBuffer` buffers via AllocMem function. Such allocations can be released via EncryptBufferFree 385 function.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
var
   Buffer: TEncryptBuffer;
begin
   // Allocate new buffer (600 bytes)
   Buffer.cbData := 600;
   Buffer.pbData := AllocMem(EncryptBuffer.cbData);
   try
      // Work with Buffer
   finally
      // Release buffer when done
      EncryptBufferFree(Buffer); // - same as SecureFree(Buffer);
   end;
end;
```

Code (Delphi)

```
var
   Buffer: TEncryptBuffer;
   S: RawByteString;
begin
   S := 'something';

   // Point buffer to S
   Buffer.cbData := Length(S);
   Buffer.pbData := Pointer(S);

   // Work with Buffer

   // No need to release Buffer
   // (we did not allocate memory for buffer)
end;
```

See also

- EncryptBufferFree Function 385

11.55 TEurekaActionType

Action kind for exception processing.

Unit

ETypes 138

Syntax

Code (Delphi)

```
TEurekaActionType = (
    atEurekaLogInitializing 675,
    atEurekaLogInitialized 675,
    atCollectingExceptionInfo 675,
    atCollectedExceptionInfo 674,
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    atExceptionHandling 676,
    atShowingExceptionInfo 677,
    atShowedExceptionInfo 677,
    atSavingLogFile 676,
    atSavedLogFile 676,
    atSending 677,
    atSent 677,
    atSendCancelled 676,
    atExceptionHandled 675,
    atTerminating 677,
    atRestarting 676,
    atEventHandler 675,
    atGeneral 676
);
```

Members

atCollectedExceptionInfo 674	[after] Fired after collecting information about exception (building call stack)
atCollectingExceptionInfo 675	[before] Fired before collecting information about exception (building call stack)
atEurekaLogInitialized 675	[after] Fired immediately after EurekaLog was initialized (ExceptionLog7 unit). Exception info will be nil. CurrentEurekaLogOptions will be available.
atEurekaLogInitializing 675	[before] Fired before EurekaLog initializes (ExceptionLog7 unit). Exception info will be nil. Do not access CurrentEurekaLogOptions from event handler
atEventHandler 675	[after] Used only in OnExceptionError event - when user event handler raises exception; Retry is ignored
atExceptionHandled 675	[after] Fired after processing exception
atExceptionHandling 676	[before] Fired before processing exception
atGeneral 676	[after] Used only in OnExceptionError event - when exception processing fails; Retry is ignored
atRestarting 676	[before] Fired before application restarting
atSavedLogFile 676	[after] Fired after saving to bug report

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

atSavingLogFile 676	[before] Fired before saving to bug report
atSendCancelled 676	[notify] Fired when sending was cancelled
atSending 677	[before] Fired before sending bug report
atSent 677	[after] Fired after sending bug report
atShowedExceptionInfo 677	[after] Fired after showing error dialog
atShowingExceptionInfo 677	[before] Fired before showing error dialog
atTerminating 677	[before] Fired before application termination

Remarks

This is type of action which is used in OnExceptionAction 16 event.

Most events can be divided into 2 groups:
- "Before-like" actions, which are fired **before** event;
- "After-like" actions, which are fired **after** event;

Those groups have different meaning in event handler with respect to `AExecute` argument.
- "Before-like" actions: `AExecute` can allow or deny action;
- "After-like" actions: `AExecute` sets execution status - success or not;

Basically, for most actions EurekaLog follow such pattern (pseudo-code):

Code (Delphi)

```
repeat
   MakeCall := DoEventExceptionAction(... "before" ...);
   if MakeCall then
      Result := PerformAction
   else
      Result := Cancelled;
   Result := DoEventExceptionAction(... "after" ...);

   if Failed(Result) then
      Retry := DoEventExceptionError(...)
    else
       Retry := False;
until not Retry;
```

See also

- Events 83
- OnExceptionAction event 16

11.55.1 atCollectedExceptionInfo

[after] Fired after collecting information about exception (building call stack)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 138

See also
- TEurekaActionType Class 672

11.55.2 atCollectingExceptionInfo

[before] Fired before collecting information about exception (building call stack)

Unit
ETypes 138

See also
- TEurekaActionType Class 672

11.55.3 atEurekaLogInitialized

[after] Fired immediately after EurekaLog was initialized (ExceptionLog7 unit). Exception info will be nil. CurrentEurekaLogOptions will be available.

Unit
ETypes 138

See also
- TEurekaActionType Class 672

11.55.4 atEurekaLogInitializing

[before] Fired before EurekaLog initializes (ExceptionLog7 unit). Exception info will be nil. Do not access CurrentEurekaLogOptions from event handler

Unit
ETypes 138

See also
- TEurekaActionType Class 672

11.55.5 atEventHandler

[after] Used only in OnExceptionError event - when user event handler raises exception; Retry is ignored

Unit
ETypes 138

See also
- TEurekaActionType Class 672

11.55.6 atExceptionHandled

Unit
ETypes 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEurekaActionType Class 672

11.55.7 atExceptionHandling

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.8 atGeneral

[after] Used only in OnExceptionError event - when exception processing fails; Retry is ignored

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.9 atRestarting

[before] Fired before application restarting

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.10 atSavedLogFile

[after] Fired after saving to bug report

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.11 atSavingLogFile

[before] Fired before saving to bug report

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.12 atSendCancelled

[notify] Fired when sending was cancelled

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.13 atSending

[before] Fired before sending bug report

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.14 atSent

[after] Fired after sending bug report

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.15 atShowedExceptionInfo

[after] Fired after showing error dialog

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.16 atShowingExceptionInfo

[before] Fired before showing error dialog

Unit

ETypes 138

See also

- TEurekaActionType Class 672

11.55.17 atTerminating

[before] Fired before application termination

Unit

ETypes 138

See also

- TEurekaActionType Class 672

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.56 TEurekaCallStack

Alias for TEurekaStackListV7.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
TEurekaCallStack = TEurekaStackList 684;
```

Remarks

This is generic name, which will be changed in next EurekaLog versions. Use this name to refer the latest possible method.

Tip

This is the same as TEurekaStackList 684.

11.57 TEurekaCallStackClass

Class of TEurekaBaseStackList.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
TEurekaCallStackClass = class of TEurekaBaseStackList;
```

11.58 TEurekaDebugDetail

Level of detalization.

Unit

ETypes 138

Syntax

Code (Delphi)

```
TEurekaDebugDetail = (
    ddNone 679,
    ddModule 679,
    ddUnit 680,
    ddProcedure 679,
    ddSourceCode 680
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

ddModule 679	Code belongs to module, but module does not have debug information, or code lies outside of debug information scope. Only module info is available.
ddNone 679	No debug information is available.
ddProcedure 679	Routine name is available.
ddSourceCode 680	Line number is available.
ddUnit 680	Unit name is available.

Remarks

Indicates what fields are filled in code location information.

See also

- TELLocationInfo Record 662
- GetLocationInfo Function 423
- TEurekaDebugDetails Type 680

11.58.1 ddModule

Code belongs to module, but module does not have debug information, or code lies outside of debug information scope. Only module info is available.

Unit

ETypes 138

See also

- TEurekaDebugDetail Class 678

11.58.2 ddNone

No debug information is available.

Unit

ETypes 138

See also

- TEurekaDebugDetail Class 678

11.58.3 ddProcedure

Routine name is available.

Unit

ETypes 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also
- TEurekaDebugDetail Class 678

11.58.4 ddSourceCode

Line number is available.

Unit
ETypes 138

See also
- TEurekaDebugDetail Class 678

11.58.5 ddUnit

Unit name is available.

Unit
ETypes 138

See also
- TEurekaDebugDetail Class 678

11.59 TEurekaDebugDetails

Set of detalization levels.

Unit
ETypes 138

Syntax

Code (Delphi)
```
TEurekaDebugDetails = set of TEurekaDebugDetail;
```

Remarks
Indicates what detalization levels should be used.

See also
- TEurekaDebugDetail Type 678
- TELLocationInfo Record 662
- GetLocationInfo Function 423

11.60 TEurekaLogFileType

Type of file name.

Unit
ETypes 138

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
TEurekaLogFileType = (
    ftSavedBugReport 682,
    ftBugReport 682,
    ftZIP 683,
    ftZIPCopy 683,
    ftXML 683,
    ftScreenshot 683,
    ftHTMLPage 682,
    ftMiniDump 682,
    ftLog 682
);
```

Members

ftBugReport 682	Actual working copy of bug report. This bug report will be used for sending (.el). Default location is the sub-folder in the TEMP folder.
ftHTMLPage 682	HTML page file name (.htm, .html). Default location is the sub-folder in the TEMP folder.
ftLog 682	Log file name (.csl). Default location is the sub-folder in the TEMP folder.
ftMiniDump 682	Minidump file name (.mdmp). Default location is the sub-folder in the TEMP folder.
ftSavedBugReport 682	Saved bug report in its final location (as set in options). E.g. local bug report (.el). Default is the sub-folder in the APPDATA folder. Can be customized in EurekaLog project options. If local bug report is disabled in options, then this value defaults to the sub-folder in the TEMP folder.
ftScreenshot 683	Screenshot file name (.png). Default location is the sub-folder in the TEMP folder.
ftXML 683	Name of XML bug report (.elx). Default location is the sub-folder in the TEMP folder.
ftZIP 683	Packed bug report for sending (ZIP/ELP, including additional files, screenshot, etc.) (.elp). Default location is the sub-folder in the TEMP folder.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ftZIPCopy 683	File name for "Save copy in case of send failure" option (.elp). Default location is the My Documents folder.

Remarks

This type is used in OnCustomFileName event handlers to specify kind of file name to modify.

11.60.1 ftBugReport

Actual working copy of bug report. This bug report will be used for sending (.el). Default location is the sub-folder in the TEMP folder.

Unit

ETypes 138

See also

- TEurekaLogFileType Class 680

11.60.2 ftHTMLPage

HTML page file name (.htm, .html). Default location is the sub-folder in the TEMP folder.

Unit

ETypes 138

See also

- TEurekaLogFileType Class 680

11.60.3 ftLog

Log file name (.csl). Default location is the sub-folder in the TEMP folder.

Unit

ETypes 138

See also

- TEurekaLogFileType Class 680

11.60.4 ftMiniDump

Minidump file name (.mdmp). Default location is the sub-folder in the TEMP folder.

Unit

ETypes 138

See also

- TEurekaLogFileType Class 680

11.60.5 ftSavedBugReport

Saved bug report in its final location (as set in options). E.g. local bug report (.el). Default is the sub-folder in the APPDATA folder. Can be customized in EurekaLog project options. If local bug report is disabled in options, then this value defaults to the sub-folder in the TEMP folder.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 138

See also
- TEurekaLogFileType Class 680

11.60.6 ftScreenshot

Screenshot file name (.png). Default location is the sub-folder in the TEMP folder.

Unit
ETypes 138

See also
- TEurekaLogFileType Class 680

11.60.7 ftXML

Name of XML bug report (.elx). Default location is the sub-folder in the TEMP folder.

Unit
ETypes 138

See also
- TEurekaLogFileType Class 680

11.60.8 ftZIP

Packed bug report for sending (ZIP/ELP, including additional files, screenshot, etc.) (.elp). Default location is the sub-folder in the TEMP folder.

Unit
ETypes 138

See also
- TEurekaLogFileType Class 680

11.60.9 ftZIPCopy

File name for "Save copy in case of send failure" option (.elp). Default location is the My Documents folder.

Unit
ETypes 138

See also
- TEurekaLogFileType Class 680

11.61 TEurekaModuleType

Alias for TModuleType 709

Unit
ETypes 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TEurekaModuleType = TModuleType 709;
```

Remarks

See description of TModuleType 709.

See also

- TModuleType

11.62 TEurekaStackList

Alias for TEurekaStackListV7.

Unit

ECallStack 59

Syntax

```
Code (Delphi)
TEurekaStackList = TEurekaStackListV7 218;
```

Remarks

This is generic name, which will be changed in next EurekaLog versions. Use this name to refer the latest possible method.

```
Tip
This is the same as TEurekaCallStack 678.
```

11.63 TExceptionNotifyMode

Specifies work mode of exception manager.

Unit

EExceptionManager 93

Syntax

```
Code (Delphi)
TExceptionNotifyMode = (
    emUnknown 686,
    emFull 686,
    emD2009 685,
    emHook 686,
    emNull 686
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

emD2009 685	Extended memory manager is not set. Hi-level hook was set for Exception class. (Available only for Delphi 2009 and later, for C++ Builder 2009 and later with classic compiler; not working in C++ Builder with LLVM compiler)
emFull 686	Extended memory manager was set. Full features.
emHook 686	Extended memory manager is not set. Low-level hook was set for TObject.Destroy. May be not compatible with exe protectors ("Use Low Level Hooks" option is enabled).
emNull 686	No additional hooks. Minimal functionality. Extended memory manager is not set. Delphi/C++ Builder version is 2007 or earlier. "Use Low Level Hooks" option is disabled or not possible to hook TObject.Destroy. This includes external (non-Delphi) exceptions.
emUnknown 686	Work mode is not set. Exception manager was not initialized yet.

Remarks

Work mode indicate what kind of additional hooks were installed by exception manager 232. Additional hooks are required for some special features to work. This includes "Catch handled exceptions" and "Handle safecall exceptions" options.

This mode doesn't indicate state of primary hooks.

See also

- TExceptionManager.Mode property 245
- TExceptionManager class 232

11.63.1 emD2009

Extended memory manager is not set. Hi-level hook was set for Exception class. (Available only for Delphi 2009 and later, for C++ Builder 2009 and later with classic compiler; not working in C++ Builder with LLVM compiler)

Unit

EExceptionManager 93

See also

- TExceptionNotifyMode Class 684

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.63.2 emFull

Extended memory manager was set. Full features.

Unit

EExceptionManager 93

See also

- TExceptionNotifyMode Class 684

11.63.3 emHook

Extended memory manager is not set. Low-level hook was set for TObject.Destroy. May be not compatible with exe protectors ("Use Low Level Hooks" option is enabled).

Unit

EExceptionManager 93

See also

- TExceptionNotifyMode Class 684

11.63.4 emNull

No additional hooks. Minimal functionality. Extended memory manager is not set. Delphi/C++ Builder version is 2007 or earlier. "Use Low Level Hooks" option is disabled or not possible to hook TObject.Destroy. This includes external (non-Delphi) exceptions.

Unit

EExceptionManager 93

See also

- TExceptionNotifyMode Class 684

11.63.5 emUnknown

Work mode is not set. Exception manager was not initialized yet.

Unit

EExceptionManager 93

See also

- TExceptionNotifyMode Class 684

11.64 TFreezeThreadClass

Class of freeze detection thread.

Unit

EFreeze 96

Syntax

Code (Delphi)

```
TFreezeThreadClass = class of TFreezeThread;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This type accepts TFreezeThread 246 class or any of its descendants.

This type is used in FreezeThreadClass 756 global variable.

Caution!

`TFreezeThread` is abstract class, you should not use it. Use TMainThreadFreezeDetectionThread 254 or TWCTFreezeDetectionThread 265 or implement your own freeze detection class.

See also

- TFreezeThread Class 246
- FreezeThreadClass Variable 756
- TMainThreadFreezeDetectionThread Class 254
- TWCTFreezeDetectionThread Class 265
- InitCheckFreeze Function 436

11.65 TInstruction

A CPU instruction.

Unit

EDisAsm 78

Syntax

Code (Delphi)

```
TInstruction = record
    Start 689: PAddress;
    Next 688: PAddress;
    Accessible 687: Boolean;
    Valid 689: Boolean;
    PlatformSpecific 688: Fields;
end;
```

Remarks

This platform-specific record represents single CPU instruction. Exact fields of the record depends on target CPU. Please, refer to source code of the unit for your CPU (such as EDisAsmX8632.pas, EDisAsmX8664.pas, etc.).

See also

- DecodeInstruction Function 353
- InstructionToString Function 441
- PInstruction Type 608

11.65.1 Accessible

Is instruction readable?

Unit

EDisAsm 78

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
Accessible: Boolean;
```

Value

`True` - instruction's memory is accessible (readable); `False` - otherwise.

Remarks

Indicates if instruction's Start 689 address is readable. You will get Access Violation exception if you try to read from `Start` when Accessible if `False`.

See also

- Start Field 689
- Next Field 688
- Valid Field 689
- TInstruction Record 687

11.65.2 Next

Next CPU instruction.

Unit

EDisAsm 78

Syntax

```
Code (Delphi)
Next: PAddress;
```

Value

Pointer to next instruction.

Remarks

This field contains pointer to end of current CPU instruction and start of the next (following) CPU instruction.

Note

You can use this field to determinate length of CPU instruction in bytes. Just subtract Start 689 from `Next`.

See also

- Start Field 689
- Accessible Field 687
- Valid Field 689
- TInstruction Record 687

11.65.3 PlatformSpecific

Other platform-specific fields.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EDisAsm 78

Syntax

Code (Delphi)

```
PlatformSpecific: Fields;
```

Value
Exact values and meaning depend on target CPU.

Remarks
This is not a real field. It is just substitute for other fields in this record. For example x86-32 CPU (implemented in `EDisAsmX8632.pas` unit) have fields like `Prefix66`, `OpCodeSignificantByte`, `OperandSize`, `ModRMPresent`, etc.

See also
- Start Field 689
- Next Field 688
- Accessible Field 687
- Valid Field 689
- TInstruction Record 687

11.65.4 Start

Start of the CPU instruction.

Unit
EDisAsm 78

Syntax

Code (Delphi)

```
Start: PAddress;
```

Value
Pointer to instruction's start.

Remarks
This field contains pointer to the start of the CPU instruction. This includes any prefixes.

See also
- Next Field 688
- Accessible Field 687
- Valid Field 689
- TInstruction Record 687

11.65.5 Valid

Is instruction valid?

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EDisAsm 78

Syntax

Code (Delphi)

```
Valid: Boolean;
```

Value
`True` - instruction was decoded successfully; `False` - unknown or invalid instruction.

Remarks
This field indicate if CPU instruction was decoded. All platform-specific fields in TInstruction record are valid only when `Valid` is `True`.

See also
- Start Field 689
- Next Field 688
- Accessible Field 687
- TInstruction Record 687

11.66 TIssueIDType

Type of the issue ID in the bug tracker.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TIssueIDType = (
   iitUnspecified,
   iitInteger,
   iitUnsigned,
   iitHex8,
   iitHex16,
   iitString,
   iitHRESULT,
   iitWin32,
   iitHTTP
);
```

Members

Remarks
This type indicates how issue ID should be formatted.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TResponse record 713

11.67 TLogMsgContent

Kind of data (content).

Unit

ELogging 103

Syntax

Code (Delphi)

```
TLogMsgContent = TLogType 692;
```

Remarks

This type specifies kind of data in the log entry (e.g. type of entry's content).

Caution!

This type is alias for TLogType 692 type. However, you can only use items starting with `elmc`.

See also

- TLogType Type 692
- TLogMsgType Type 691

11.68 TLogMsgType

Kind of log entry.

Unit

ELogging 103

Syntax

Code (Delphi)

```
TLogMsgType = TLogType 692;
```

Remarks

This type specifies kind of single log entry.

Caution!

This type is alias for TLogType 692 type. However, you can only use items starting with `elmt`.

See also

- TLogType Type 692
- TLogMsgContent Type 691
- IEurekaLogger.Kind Method 284

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69 TLogType

Common "type".

Unit

ELogging 103

Syntax

Code (Delphi)

```
TLogType = (
    elmtInfo 701,
    elmtWarning 704,
    elmtError 700,
    elmtCheckPoint 699,
    elmcObject 697,
    elmcComponentAsText 696,
    elmcStringList 697,
    elmcText 698,
    elmtNote 703,
    elmtEnterMethod 700,
    elmtExitMethod 700,
    elmtSeparator 704,
    elmtInactiveSeparator 701,
    elmtClear 699,
    elmtScratchPad 704,
    elmtSaveLogFile 703,
    elmcBitmap 695,
    elmcTextFile 698,
    elmcWindow 698,
    elmcMemory 697,
    elmcMemoryStatus 697,
    elmcCustomText 696,
    elmcCustomProperties 696,
    elmtReminder 703,
    elmtLevel1 701,
    elmtLevel2 701,
    elmtLevel3 702,
    elmtLevel4 702,
    elmtLevel5 702,
    elmtLevel6 702,
    elmtLevel7 702,
    elmcColor 695,
    elmcRegistry 697,
    elmcSystemInfo 698,
    elmcVersionInfo 698,
    elmcXmlData 699,
    elmcStackTrace 697,
    elmtException 700,
    elmcCustomTable 696,
    elmtResetSeparator 703,
    elmtAppStartSeparator 699,
    elmcCollection 695,
    elmtCollapsedMethod 699,
    elmtExitMethodCollapse 700,
```


© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    elmcCustomHierarchy 696,
    elmtTiming 704,
    elmtRed 703,
    elmtOrange 703,
    elmtYellow 705,
    elmtGreen 701,
    elmtBlue 699,
    elmtIndigo 701,
    elmtViolet 704
);
```

Members

elmcBitmap 695	[TLogMsgContent] csmBitmap
elmcCollection 695	[TLogMsgContent] csmCollection
elmcColor 695	[TLogMsgContent] csmColor
elmcComponentAsText 696	[TLogMsgContent] csmComponentAsText
elmcCustomHierarchy 696	[TLogMsgContent] csmCustomHierarchy
elmcCustomProperties 696	[TLogMsgContent] csmCustomProperties
elmcCustomTable 696	[TLogMsgContent] csmCustomTable
elmcCustomText 696	[TLogMsgContent] csmCustomText
elmcMemory 697	[TLogMsgContent] csmMemory
elmcMemoryStatus 697	[TLogMsgContent] csmMemoryStatus
elmcObject 697	[TLogMsgContent] csmObject
elmcRegistry 697	[TLogMsgContent] csmRegistry
elmcStackTrace 697	[TLogMsgContent] csmStackTrace
elmcStringList 697	[TLogMsgContent] csmStringList
elmcSystemInfo 698	[TLogMsgContent] csmSystemInfo
elmcText 698	[TLogMsgContent] csmText
elmcTextFile 698	[TLogMsgContent] csmTextFile
elmcVersionInfo 698	[TLogMsgContent] csmVersionInfo

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

elmcWindow 698	[TLogMsgContent] csmWindow
elmcXmlData 699	[TLogMsgContent] csmXmlData
elmtAppStartSeparator 699	[TLogMsgType] csmAppStartSeparator
elmtBlue 699	[TLogMsgType] csmBlue
elmtCheckPoint 699	[TLogMsgType] csmCheckPoint
elmtClear 699	[TLogMsgType] csmClear
elmtCollapsedMethod 699	[TLogMsgType] csmCollapsedMethod
elmtEnterMethod 700	[TLogMsgType] csmEnterMethod
elmtError 700	[TLogMsgType] csmError
elmtException 700	[TLogMsgType] csmException
elmtExitMethod 700	[TLogMsgType] csmExitMethod
elmtExitMethodCollapse 700	[TLogMsgType] csmExitMethodCollapse
elmtGreen 701	[TLogMsgType] csmGreen
elmtInactiveSeparator 701	[TLogMsgType] csmInactiveSeparator
elmtIndigo 701	[TLogMsgType] csmIndigo
elmtInfo 701	[TLogMsgType] csmInfo
elmtLevel1 701	[TLogMsgType] csmLevel1
elmtLevel2 701	[TLogMsgType] csmLevel2
elmtLevel3 702	[TLogMsgType] csmLevel3
elmtLevel4 702	[TLogMsgType] csmLevel4
elmtLevel5 702	[TLogMsgType] csmLevel5
elmtLevel6 702	[TLogMsgType] csmLevel6
elmtLevel7 702	[TLogMsgType] csmLevel7
elmtNote 703	[TLogMsgType] csmNote
elmtOrange 703	[TLogMsgType] csmOrange

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

elmtRed 703	[TLogMsgType] csmRed
elmtReminder 703	[TLogMsgType] csmReminder
elmtResetSeparator 703	[TLogMsgType] csmResetSeparator
elmtSaveLogFile 703	[TLogMsgType] csmSaveLogFile
elmtScratchPad 704	[TLogMsgType] csmScratchPad
elmtSeparator 704	[TLogMsgType] csmSeparator
elmtTiming 704	[TLogMsgType] csmTiming
elmtViolet 704	[TLogMsgType] csmViolet
elmtWarning 704	[TLogMsgType] csmWarning
elmtYellow 705	[TLogMsgType] csmYellow

Remarks

This type is a base type for all "types" and "kinds" in the log file. It is compatible with CodeSite's csmXYZ consts from CodeSiteMsgTypes.inc unit.

See also

- TLogMsgType Type 691
- TLogMsgContent Type 691

11.69.1 elmcBitmap

[TLogMsgContent] csmBitmap

Unit

ELogging 103

See also

- TLogType Class 692

11.69.2 elmcCollection

[TLogMsgContent] csmCollection

Unit

ELogging 103

See also

- TLogType Class 692

11.69.3 elmcColor

[TLogMsgContent] csmColor

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ELogging 103

See also

- TLogType Class 692

11.69.4 elmcComponentAsText

[TLogMsgContent] csmComponentAsText

Unit

ELogging 103

See also

- TLogType Class 692

11.69.5 elmcCustomHierarchy

[TLogMsgContent] csmCustomHierarchy

Unit

ELogging 103

See also

- TLogType Class 692

11.69.6 elmcCustomProperties

[TLogMsgContent] csmCustomProperties

Unit

ELogging 103

See also

- TLogType Class 692

11.69.7 elmcCustomTable

[TLogMsgContent] csmCustomTable

Unit

ELogging 103

See also

- TLogType Class 692

11.69.8 elmcCustomText

[TLogMsgContent] csmCustomText

Unit

ELogging 103

See also

- TLogType Class 692

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69.9 elmcMemory

[TLogMsgContent] csmMemory

Unit
ELogging 103

See also
- TLogType Class 692

11.69.10 elmcMemoryStatus

[TLogMsgContent] csmMemoryStatus

Unit
ELogging 103

See also
- TLogType Class 692

11.69.11 elmcObject

[TLogMsgContent] csmObject

Unit
ELogging 103

See also
- TLogType Class 692

11.69.12 elmcRegistry

[TLogMsgContent] csmRegistry

Unit
ELogging 103

See also
- TLogType Class 692

11.69.13 elmcStackTrace

[TLogMsgContent] csmStackTrace

Unit
ELogging 103

See also
- TLogType Class 692

11.69.14 elmcStringList

[TLogMsgContent] csmStringList

Unit
ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TLogType Class 692

11.69.1.5 elmcSystemInfo

[TLogMsgContent] csmSystemInfo

Unit

ELogging 103

See also

- TLogType Class 692

11.69.1.6 elmcText

[TLogMsgContent] csmText

Unit

ELogging 103

See also

- TLogType Class 692

11.69.1.7 elmcTextFile

[TLogMsgContent] csmTextFile

Unit

ELogging 103

See also

- TLogType Class 692

11.69.1.8 elmcVersionInfo

[TLogMsgContent] csmVersionInfo

Unit

ELogging 103

See also

- TLogType Class 692

11.69.1.9 elmcWindow

[TLogMsgContent] csmWindow

Unit

ELogging 103

See also

- TLogType Class 692

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69.2(elmcXmlData

[TLogMsgContent] csmXmlData

Unit
ELogging 103

See also
- TLogType Class 692

11.69.2!elmtAppStartSeparator

[TLogMsgType] csmAppStartSeparator

Unit
ELogging 103

See also
- TLogType Class 692

11.69.2"elmtBlue

[TLogMsgType] csmBlue

Unit
ELogging 103

See also
- TLogType Class 692

11.69.2#elmtCheckPoint

[TLogMsgType] csmCheckPoint

Unit
ELogging 103

See also
- TLogType Class 692

11.69.2$elmtClear

[TLogMsgType] csmClear

Unit
ELogging 103

See also
- TLogType Class 692

11.69.2%elmtCollapsedMethod

[TLogMsgType] csmCollapsedMethod

Unit
ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TLogType Class 692

11.69.26 elmtEnterMethod

[TLogMsgType] csmEnterMethod

Unit

ELogging 103

See also

- TLogType Class 692

11.69.27 elmtError

[TLogMsgType] csmError

Unit

ELogging 103

See also

- TLogType Class 692

11.69.28 elmtException

[TLogMsgType] csmException

Unit

ELogging 103

See also

- TLogType Class 692

11.69.29 elmtExitMethod

[TLogMsgType] csmExitMethod

Unit

ELogging 103

See also

- TLogType Class 692

11.69.30 elmtExitMethodCollapse

[TLogMsgType] csmExitMethodCollapse

Unit

ELogging 103

See also

- TLogType Class 692

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69.31 elmtGreen

[TLogMsgType] csmGreen

Unit
ELogging 103

See also
- TLogType Class 692

11.69.32 elmtInactiveSeparator

[TLogMsgType] csmInactiveSeparator

Unit
ELogging 103

See also
- TLogType Class 692

11.69.33 elmtIndigo

[TLogMsgType] csmIndigo

Unit
ELogging 103

See also
- TLogType Class 692

11.69.34 elmtInfo

[TLogMsgType] csmInfo

Unit
ELogging 103

See also
- TLogType Class 692

11.69.35 elmtLevel1

[TLogMsgType] csmLevel1

Unit
ELogging 103

See also
- TLogType Class 692

11.69.36 elmtLevel2

[TLogMsgType] csmLevel2

Unit
ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TLogType Class 692

11.69.37 elmtLevel3

[TLogMsgType] csmLevel3

Unit

ELogging 103

See also

- TLogType Class 692

11.69.38 elmtLevel4

[TLogMsgType] csmLevel4

Unit

ELogging 103

See also

- TLogType Class 692

11.69.39 elmtLevel5

[TLogMsgType] csmLevel5

Unit

ELogging 103

See also

- TLogType Class 692

11.69.40 elmtLevel6

[TLogMsgType] csmLevel6

Unit

ELogging 103

See also

- TLogType Class 692

11.69.41 elmtLevel7

[TLogMsgType] csmLevel7

Unit

ELogging 103

See also

- TLogType Class 692

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69.4.2 elmtNote

[TLogMsgType] csmNote

Unit
ELogging 103

See also
- TLogType Class 692

11.69.4.3 elmtOrange

[TLogMsgType] csmOrange

Unit
ELogging 103

See also
- TLogType Class 692

11.69.4.4 elmtRed

[TLogMsgType] csmRed

Unit
ELogging 103

See also
- TLogType Class 692

11.69.4.5 elmtReminder

[TLogMsgType] csmReminder

Unit
ELogging 103

See also
- TLogType Class 692

11.69.4.6 elmtResetSeparator

[TLogMsgType] csmResetSeparator

Unit
ELogging 103

See also
- TLogType Class 692

11.69.4.7 elmtSaveLogFile

[TLogMsgType] csmSaveLogFile

Unit
ELogging 103

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TLogType Class 692

11.69.4 elmtScratchPad

[TLogMsgType] csmScratchPad

Unit

ELogging 103

See also

- TLogType Class 692

11.69.4 elmtSeparator

[TLogMsgType] csmSeparator

Unit

ELogging 103

See also

- TLogType Class 692

11.69.5 elmtTiming

[TLogMsgType] csmTiming

Unit

ELogging 103

See also

- TLogType Class 692

11.69.5 elmtViolet

[TLogMsgType] csmViolet

Unit

ELogging 103

See also

- TLogType Class 692

11.69.5 elmtWarning

[TLogMsgType] csmWarning

Unit

ELogging 103

See also

- TLogType Class 692

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.69.5 lmtYellow

[TLogMsgType] csmYellow

Unit
ELogging 103

See also
- TLogType Class 692

11.70 TMD5Hash

MD5 hash.

Unit
EHash 97

Syntax

Code (Delphi)

```
TMD5Hash = packed array[0..15]of Byte;
```

Remarks
Data type to hold MD5 hash. This is 16 bytes (128 bits) value.

See also
- MD5Hash Function 480

11.71 TMemoryError

Type of detected memory error.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TMemoryError = (
    leNone 707,
    leOverflow 708,
    leMultiFree 707,
    leObjectInUse 707,
    leInvalidBlock 707,
    leHang 706,
    leUseAfterFree 708,
    leOutOfMemory 707,
    leUninstalled 708
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

leHang 706	Memory manager is unresponsive, e.g. hang, deadlocked, etc. This error can happen due to damaged internal structures of memory manager, terminated thread, etc.
leInvalidBlock 707	A pointer to invalid block was passed to FreeMem or ReallocMem. Invalid block is a block which does not contain a valid EurekaLog header. This can be a valid, but corrupted block (overflow/ underflow), it can be already released block (double-free), it can be a fully rogue/random pointer, it can be a block allocated by another memory manager.
leMultiFree 707	Double-free of memory block or object.
leNone 707	No memory error is detected.
leObjectInUse 707	attempt to delete object, which is still in use
leOutOfMemory 707	There is not enough memory to allocate requested amount of memory in GetMem or AllocMem.
leOverflow 708	Buffer overflow is detected. Some code damaged/overwrote internal EurekaLog's headers.
leUninstalled 708	Application made attempt to call memory manager's function after finalization. This usually happens when using shared memory manager and wrong finalization order for DLLs.
leUseAfterFree 708	Detected attempt to use already released object or interface. A virtual method was called on already deleted object.

Remarks

This type is used in TMemLeaksError event handler.

See also

- TMemLeaksError Type
- MemLeaksError Global Variable

11.71.1 leHang

Memory manager is unresponsive, e.g. hang, deadlocked, etc. This error can happen due to damaged internal structures of memory manager, terminated thread, etc.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ETypes 138

See also

- TMemoryError Class 705

11.71.2 leInvalidBlock

A pointer to invalid block was passed to FreeMem or ReallocMem. Invalid block is a block which does not contain a valid EurekaLog header. This can be a valid, but corrupted block (overflow/underflow), it can be already released block (double-free), it can be a fully rogue/random pointer, it can be a block allocated by another memory manager.

Unit

ETypes 138

See also

- TMemoryError Class 705

11.71.3 leMultiFree

Double-free of memory block or object.

Unit

ETypes 138

See also

- TMemoryError Class 705

11.71.4 leNone

No memory error is detected.

Unit

ETypes 138

See also

- TMemoryError Class 705

11.71.5 leObjectInUse

attempt to delete object, which is still in use

Unit

ETypes 138

See also

- TMemoryError Class 705

11.71.6 leOutOfMemory

There is not enough memory to allocate requested amount of memory in GetMem or AllocMem.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 138

See also
- TMemoryError Class 705

11.71.7 leOverflow

Buffer overflow is detected. Some code damaged/overwrote internal EurekaLog's headers.

Unit
ETypes 138

See also
- TMemoryError Class 705

11.71.8 leUninstalled

Application made attempt to call memory manager's function after finalization. This usually happens when using shared memory manager and wrong finalization order for DLLs.

Unit
ETypes 138

See also
- TMemoryError Class 705

11.71.9 leUseAfterFree

Detected attempt to use already released object or interface. A virtual method was called on already deleted object.

Unit
ETypes 138

See also
- TMemoryError Class 705

11.72 TMemoryErrors

Defines set of possible memory errors.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TMemoryErrors = set of TMemoryError;
```

Remarks
This type is used in MemLeaksErrorsToIgnore global variable.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TMemoryError Type
- MemLeaksErrorsToIgnore Global Variable

11.73 TModuleType

Indicates kind of executable module.

Unit

ETypes 138

Syntax

Code (Delphi)

```
TModuleType = (
    mtUnknown 710,
    mtProgram 710,
    mtLibrary 709,
    mtPackage 710
);
```

Members

mtLibrary 709	.dll file
mtPackage 710	.bpl file
mtProgram 710	.exe file
mtUnknown 710	Module type wasn't recognized

Remarks

Module type is used to draw call stack items and for design/compiler part of EurekaLog.

See also

- GetModuleType Function 425
- FindModuleType Method
- TELLocationInfo Type 662

11.73.1 mtLibrary

.dll file

Unit

ETypes 138

See also

- TModuleType Class 709

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.73.2 mtPackage

.bpl file

Unit
ETypes 138

See also
- TModuleType Class 709

11.73.3 mtProgram

.exe file

Unit
ETypes 138

See also
- TModuleType Class 709

11.73.4 mtUnknown

Module type wasn't recognized

Unit
ETypes 138

See also
- TModuleType Class 709

11.74 TPointerRecord

Header for memory blocks allocated by EurekaLog.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TPointerRecord = packed record
   Prev, Next: PPointerRecord 608;
   Module: HMODULE;
   Tag: Pointer;
   OrigSize: PtrUInt 609;
   SingleSize 712: PtrUInt 609;
   TotalSize: PtrUInt 609;
   Name: TCustomName 622;
   Parent: Pointer;
   BlockState 711: TBlockState;
   BlockType 711: TBlockType 611;
   Reserved1: WORD;
   ThreadID: DWORD;
   Count: DWORD;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    Flags: DWORD;
    CallStack: TCallStackArray;
    GuardBlock: DWord;
end;
```

Remarks

This data structure represents internal EurekaLog header for allocated memory blocks. Memory block's data starts immediately after this structure.

Warning
This data type is subject to change. Consider it to be undocumented.

See also

- DumpAllocationsToFile Function
- TDumpCompareFunc Type
- TDumpFilterFunc Type
- PPointerRecord Type

11.74.1 BlockState

State of memory block.

Unit

ETypes

Syntax

Code (Delphi)

```
BlockState: TBlockState;
```

Value

Indicates state of the block.

Remarks

This value represents state (allocated, released, filtered, etc.) of the memory block.

See also

- TPointerRecord Record

11.74.2 BlockType

Type of memory block.

Unit

ETypes

Syntax

Code (Delphi)

```
BlockType: TBlockType;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
Indicate type of the block.

Remarks
This value represents type of memory block (object, array, string or RAW).

See also
- TPointerRecord Record 710

11.74.3 CallStack

Call stack of allocation.

Unit
ETypes 138

Syntax

Code (Delphi)
```
CallStack: TCallStackArray;
```

Value
Array of RAW pointers representing return addresses.

Remarks
This is call stack taken when memory block was first allocated.

Tip

Call stack holds array of return addresses. Return address is not the same as pointer to call instruction.

See also
- TPointerRecord Record 710

11.74.4 SingleSize

Memory block size.

Unit
ETypes 138

Syntax

Code (Delphi)
```
SingleSize: PtrUInt 609;
```

Value
Actual size of the block, in bytes.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This value represents size of the memory block in bytes.

See also

- TPointerRecord Record 710

11.75 TResponse

Result of exception's processing or sending bug report.

Unit

ETypes 138

Syntax

Code (Delphi)

```
TResponse = packed record
    SendResult 716: TSendResult 722;
    IssueID 714: Int64;
    IssueIDStr 715: String;
    IssueIDType 715: TIssueIDType 690;
    ErrorCode 713: Integer;
    ErrorMessage 714: String;
    SuccessMessage 716: String;
end;
```

Remarks

This record is used to hold result of processing or sending. There are fields for status, codes, messages and ticket (issue) ID.

This data type is returned by all EurekaLog methods and functions which process exceptions.

See also

- TExceptionManager.Handle 234

11.75.1 ErrorCode

Error code for failed operations.

Unit

ETypes 138

Syntax

Code (Delphi)

```
ErrorCode: Integer;
```

Value

System or any other error code. Numeric value. May be 0, if no error code available.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Optional error code associated with failed operations. The exact meaning of error code is not defined. It can be system error code, WinINet error code, etc.

The only thing that you can do with this field - display it to user.

Caution!

This value can be 0 for failed operations. Do not assume success/failure status of operation based on this field. Use SendResult field to determinate generic state of operation.

See also

- TResponse Record 713

11.75.2 ErrorMessage

Error message (failed operations only).

Unit

ETypes 138

Syntax

Code (Delphi)

```
ErrorMessage: String;
```

Value

Error message in arbitrary format.

Remarks

Optional error message which describes occurred error in human-readable form.

This value is supposed to be shown to end-user.

See also

- TResponse Record 713

11.75.3 IssueID

Ticket/issue ID in bug tracker (number).

Unit

ETypes 138

Syntax

Code (Delphi)

```
IssueID: Int64;
```

Value

ID of processed exception in bug tracker.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This field is set only if exception report was sent to bug tracker which supports feedback (allow to retrieve ID).

ID is stored in format of bug tracker. The `IssueIDType` field indicate type of the ID.

This value can be set only for succeeded operations.

See also

- TResponse Record 713

11.75.4 IssueIDStr

Ticket/issue ID in bug tracker (string).

Unit

ETypes 138

Syntax

Code (Delphi)

```
IssueIDStr: String;
```

Value

ID of processed exception in bug tracker.

Remarks

This field is set only if exception report was sent to bug tracker which supports feedback (allow to retrieve ID).

ID is stored in format of bug tracker. The `IssueIDType` field indicate type of the ID.

This value can be set only for succeeded operations.

See also

- TResponse Record 713

11.75.5 IssueIDType

Type of the `IssueID` field.

Unit

ETypes 138

Syntax

Code (Delphi)

```
IssueIDType: TIssueIDType 690;
```

Value

ID's type of processed exception in bug tracker.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This field is set only if exception report was sent to bug tracker which supports feedback (allow to retrieve ID).

This value can be set only for succeeded operations.

See also

- TResponse Record 713

11.75.6 SendResult

Status of operation.

Unit

ETypes 138

Syntax

Code (Delphi)

```
SendResult: TSendResult 722;
```

Value

HRESULT value indicating success/failure status of operation. See TSendResult 722 for more information.

Remarks

This value indicate result of exception's processing. There are 3 success codes (Send/BugClosed/NoExceptionInfo) and multiple error (failure) codes.

Success codes indicate that exception was processed successfully. "No exception to process/handle" is also considered as success.

Failure codes indicate that there was some kind of a error during exception processing. Restart and cancelling are considered as failures (however, often there are special cases to handle these codes).

See also

- TSendResult 722
- TResponse Record 713

11.75.7 SuccessMessage

Success message (succeeded operations only).

Unit

ETypes 138

Syntax

Code (Delphi)

```
SuccessMessage: String;
```

Value

Success message in arbitrary format. This message may also be a URL.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This field may contain optional "success" message, which is supposed to be shown to end-user. Typically, this value is set by send method which works with bug trackers. Bug trackers may report status of bug (open/ closed), instructions to do ("please, download update - it fixes this issue"), and URL to open.

This field may contain either human-readable message or URL.

Human-readable message must be shown to end-user in a message box (or via similar feature in your application).

URLs must be opened in default application.

You can determinate kind of string by checking beginning of the string: does it start with URL protocol name? Currently EurekaLog is able to recognize HTTP/HTTPS protocols and automatically launch web-browser when "Show success message" option is used.

See also

- TResponse Record 713

11.76 TRSAExport

Defines how to export/import RSA key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
TRSAExport = (
    rsBLOB 718,
    rsDER 718,
    rsPEM 718
);
```

Members

rsBLOB 718	Export/import in RAW BLOB with header (Microsoft PUBLICKEYBLOB/ PRIVATEKEYBLOB, little-endian byte order).
rsDER 718	Export/import in RAW ASN.1-encoded form (DER), big-endian. This format corresponds to RSA_CSP_PUBLICKEYBLOB/ PKCS_RSA_PRIVATE_KEY, and has NO optional header.
rsPEM 718	Export/import in text form (PEM=base64-encoded DER).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- RSALoadPublicKey 533
- RSALoadPrivateKey 532
- RSASavePublicKey 535
- RSASavePrivateKey 534

11.76.1 rsBLOB

Export/import in RAW BLOB with header (Microsoft PUBLICKEYBLOB/PRIVATEKEYBLOB, little-endian byte order).

Unit

EEncrypt 80

See also

- TRSAExport Class 717

11.76.2 rsDER

Export/import in RAW ASN.1-encoded form (DER), big-endian. This format corresponds to RSA_CSP_PUBLICKEYBLOB/PKCS_RSA_PRIVATE_KEY, and has NO optional header.

Unit

EEncrypt 80

See also

- TRSAExport Class 717

11.76.3 rsPEM

Export/import in text form (PEM = base64-encoded DER).

Unit

EEncrypt 80

See also

- TRSAExport Class 717

11.77 TRSAImport

Alias to `TRSAExport`.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
TRSAImport = TRSAExport 717;
```

See also

- TRSAExport 717

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.78 TRSAKey

Key for RSA encryption.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
TRSAKey = record
    PublicKey 720: TEncryptBuffer 671;
    PrivateKey 719: TEncryptBuffer 671;
end;
```

Remarks

This is private and public keys pair used for asymmetrical encryption, decryption, signing and verification by using RSA cipher.

You need to free this key after you have done working with it - by using RSADestroyKey 527 function.

This key can be created by RSAGenKey 531 function or be imported by RSALoadPublicKey 533 and RSALoadPrivateKey 532 functions. You need to be careful when handling private key part of this key. I.e. leaking private key means leaking your key.

Use RSAEncrypt 528 function to encrypt (by using your public key) "clear text" data, use RSADecrypt 524 function to decrypt (by using your private key) encrypted data and get "clear text" back.

Use RSASign 536 function to sign (by using your private key) your data, use RSAVerify 538 to verify (by using your public key) a signature.

See also

- RSAEncrypt Function 528
- RSADecrypt Function 524
- RSASign Function 536
- RSAVerify Function 538
- RSALoadPublicKey Function 533
- RSALoadPrivateKey Function 532
- RSADestroyKey Function 527

11.78.1 PrivateKey

Contains private key part of RSA key.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
PrivateKey: TEncryptBuffer 671;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
RAW binary data of encrypted private key.

Remarks
This field holds protected private key in encrypted form. The clean unencrypted/unprotected form of private key is not available.

Use RSASavePrivateKey 534 function to export this value. Do not copy this value directly, as it will be unusable.

See also
- RSASavePrivateKey Function 534
- TRSAKey Record 719

11.78.2 PublicKey

Contains public key part of RSA key.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
PublicKey: TEncryptBuffer 671;
```

Value
RAW binary data of public key.

Remarks
This field holds unprotected public key in clear form.

Use RSASavePublicKey 535 function to export this value or simply copy this value directly.

See also
- RSASavePublicKey Function 535
- TRSAKey Record 719

11.79 TRSASignature

Digital signature.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
TRSASignature = TEncryptBuffer 671;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This type holds RAW bytes of RSA digital signature. This data can be freely copied or exported/imported as RAW bytes. No encoding, encryption/decryption is required.

Use SecureFree 545 function to release this type.

See also

- RSASign Function 536
- RSAVerify Function 538
- TEncryptBuffer Type 671

11.80 TSalt

Salt value for deriving key from password.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
TSalt = TTEAKey 737;
```

Remarks

Use InitSalt 438 to obtain new random salt. Pass the obtained value to TEADeriveKey 565 function or TwofishDeriveKey 580 function.

Note

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

See also

- PSalt Type 609
- TEADeriveKey Function 565
- TwofishDeriveKey Function 580
- InitSalt Function 438

11.81 TScreeenshotHandle

Screenshot descriptor.

Unit

EScreenShot 119

Syntax

Code (Delphi)

```
TScreeenshotHandle = type Pointer;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This value is created by a call to the ScreenshotInit 543 routine and must be released by a call to the ScreenshotFree 543 routine.

It can also be passed to the ScreenshotSave 544 routine.

See also

- ScreenshotInit Routine 543
- ScreenshotSave Routine 544
- ScreenshotFree Routine 543

11.82 TSDBMHash

SDBM hash.

Unit

EHash 97

Syntax

Code (Delphi)

```
TSDBMHash = type Cardinal;
```

Remarks

Data type to hold SDBM hash. This is 4 byte (32 bit) value (Cardinal).

See also

- SDBMHash Function 545

11.83 TSendResult

Result kind for exception processing.

Unit

ETypes 138

Syntax

Code (Delphi)

```
TSendResult = (
    srSent 727 = S_OK,
    srBugClosed 724 = S_FALSE,
    srBugReopen 725 = $00000002,
    srNoExceptionInfo 726 = S_NONE,
    srRestart 727 = E_RESTART,
    srCancelled 725 = E_CANCELLED,
    srUnknownError 727 = E_UNEXPECTED,
    srUnsupportedSender 727 = E_NOINTERFACE,
    srConnectionError 725 = E_CONNECTION_REFUSED,
    srObsoleteVersion 726 = E_OBSOLETE,
    srInvalidLogin 725 = E_LOGON_FAILED,
    srInvalidSearch 726 = E_INVALID_SEARCH,
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
    srInvalidSelection 726 = E_INVALID_SELECTION,
    srInvalidRead 726 = E_INVALID_READ,
    srInvalidInsert 725 = E_INVALID_INSERT,
    srInvalidModify 726 = E_INVALID_MODIFY
);
```

Members

srBugClosed 724	[SUCCESS] Exception was processed without errors. Sending method is set up, bug report was sent succcessfully and (most importantly) send method discovered that this exception is marked as "Fixed" in the target bug tracker.
srBugReopen 725	[SUCCESS] Exception was processed without errors. Sending method is set up, bug report was sent succcessfully and (most importantly) send method discovered that this exception is marked as "Fixed" in the target bug tracker, but the report was send anyway (ticket was reopened).
srCancelled 725	[FAILURE] Exception processing was cancelled by user via dialog.
srConnectionError 725	[FAILURE] Network connection error. This includes DNS errors.
srInvalidInsert 725	[FAILURE] There was error when creating (opening) new case (ticket). For non-bug trackers methods this means error during content sending.
srInvalidLogin 725	[FAILURE] There was error during log in stage. Most common reason is wrong login/password pair.
srInvalidModify 726	[FAILURE] There was error when modifying existing case (ticket). Applicable only for bug trackers.
srInvalidRead 726	[FAILURE] There was error during reading project or case options. Possible access denied error?
srInvalidSearch 726	[FAILURE] There was error during project search. Reasons may be access denied or project was removed. Typically this is applicable only for bug trackers.
srInvalidSelection 726	[FAILURE] There was error during selecting project (making project active).

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

	This value is used rarely, most often it's set by web-browser access methods.
srNoExceptionInfo 726	[SUCCESS] There is no active exception. Nothing to do.
srObsoleteVersion 726	[FAILURE] Project version is obsolete. Bug reports are no longer required, submission is closed.
srRestart 727	[FAILURE] Exception processing must be restarted. This value is typically used to switch between dialogs and handlers.
srSent 727	[SUCCESS] Exception was processed without errors. If sending is set up - then bug report was sent succcessfully.
srUnknownError 727	[FAILURE] Common error code. It's used when no other code can be used.
srUnsupportedSender 727	[FAILURE] Unknown sender class. It's either wrong class name specified or sender class wasn't registered.

Remarks

This value indicate result of exception's processing. There are 3 success codes (Send/BugClosed/ NoExceptionInfo) and multiple error (failure) codes.

Success codes indicate that exception was processed successfully. "No exception to process/handle" is also considered as success.

Failure codes indicate that there was some kind of a error during exception processing. Restart and cancelling are considered as failures (however, often there are special cases to handle these codes).

UnknownError code is most general failure code. It's used when other codes can not be used.

See also

- TExceptionManager.Handle 234

11.83.1 srBugClosed

[SUCCESS] Exception was processed without errors. Sending method is set up, bug report was sent succcessfully and (most importantly) send method discovered that this exception is marked as "Fixed" in the target bug tracker.

Unit

ETypes 138

See also

- TSendResult Class 722

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.83.2 srBugReopen

[SUCCESS] Exception was processed without errors. Sending method is set up, bug report was sent succcessfully and (most importantly) send method discovered that this exception is marked as "Fixed" in the target bug tracker, but the report was send anyway (ticket was reopened).

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.3 srCancelled

[FAILURE] Exception processing was cancelled by user via dialog.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.4 srConnectionError

[FAILURE] Network connection error. This includes DNS errors.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.5 srInvalidInsert

[FAILURE] There was error when creating (opening) new case (ticket). For non-bug trackers methods this means error during content sending.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.6 srInvalidLogin

[FAILURE] There was error during log in stage. Most common reason is wrong login/password pair.

Unit

ETypes 138

See also

- TSendResult Class 722

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.83.7 srInvalidModify

[FAILURE] There was error when modifying existing case (ticket). Applicable only for bug trackers.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.8 srInvalidRead

[FAILURE] There was error during reading project or case options. Possible access denied error?

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.9 srInvalidSearch

[FAILURE] There was error during project search. Reasons may be access denied or project was removed. Typically this is applicable only for bug trackers.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.10 srInvalidSelection

[FAILURE] There was error during selecting project (making project active). This value is used rarely, most often it's set by web-browser access methods.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.11 srNoExceptionInfo

[SUCCESS] There is no active exception. Nothing to do.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.12 srObsoleteVersion

[FAILURE] Project version is obsolete. Bug reports are no longer required, submission is closed.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.13 srRestart

[FAILURE] Exception processing must be restarted. This value is typically used to switch between dialogs and handlers.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.14 srSent

[SUCCESS] Exception was processed without errors. If sending is set up - then bug report was sent succcessfully.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.15 srUnknownError

[FAILURE] Common error code. It's used when no other code can be used.

Unit

ETypes 138

See also

- TSendResult Class 722

11.83.16 srUnsupportedSender

[FAILURE] Unknown sender class. It's either wrong class name specified or sender class wasn't registered.

Unit

ETypes 138

See also

- TSendResult Class 722

11.84 TSendScreenshot

Specifies screen area.

Unit

ETypes 138

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TSendScreenshot = (
    ssNone 729,
    ssForegroundWindow 729,
    ssActiveWindow 729,
    ssApplication 729,
    ssWorkarea 730,
    ssPrimary 730,
    ssDesktop 729
);
```

Members

ssActiveWindow 729	Screen area occupied by window returned by GetActiveWindow.
ssApplication 729	Screen area occupied by all windows belonging to current application. E.g. bounding rectangle of all application's windows.
ssDesktop 729	Full virtual screen. The virtual screen is the bounding rectangle of all display monitors.
ssForegroundWindow 729	Screen area occupied by window returned by GetForegroundWindow.
ssNone 729	Do not take screenshot.
ssPrimary 730	Screen area of primary display monitor.
ssWorkarea 730	Work area on the primary display monitor. The work area is the portion of the screen not obscured by the system taskbar or by application desktop toolbars.

Remarks

This type indicate what screen area should be included for the screenshot.

ssNone disables screenshot.

ssDesktop captures full screenshot.

Other values captures only some part.

See also

- FindScreenshotRect Routine 400
- ScreenshotInit Routine 543

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- Screenshot Routine 541

11.84.1 ssActiveWindow

Screen area occupied by window returned by GetActiveWindow.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.84.2 ssApplication

Screen area occupied by all windows belonging to current application. E.g. bounding rectangle of all application's windows.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.84.3 ssDesktop

Full virtual screen. The virtual screen is the bounding rectangle of all display monitors.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.84.4 ssForegroundWindow

Screen area occupied by window returned by GetForegroundWindow.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.84.5 ssNone

Do not take screenshot.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.84.6 ssPrimary

Screen area of primary display monitor.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.84.7 ssWorkarea

Work area on the primary display monitor. The work area is the portion of the screen not obscured by the system taskbar or by application desktop toolbars.

Unit
ETypes 138

See also
- TSendScreenshot Class 727

11.85 TSHA1Hash

SHA-1 hash.

Unit
EHash 97

Syntax

Code (Delphi)
```
TSHA1Hash = packed array[0..19]of Byte;
```

Remarks
Data type to hold SHA-1 hash. This is 20 bytes (160 bits) value.

See also
- SHA1Hash Function 553

11.86 TSHA256Hash

SHA-2/SHA-256 hash.

Unit
EHash 97

Syntax

Code (Delphi)
```
TSHA256Hash = packed array[0..31]of Byte;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Data type to hold SHA-2/SHA-256 hash. This is 32 bytes (256 bits) value.

See also

- SHA256Hash Function 555

11.87 TSignatureInfo

Information about digital signature of executable file.

Unit

ESignature 131

Syntax

Code (Delphi)

```
TSignatureInfo = record
    FileName 732: String;
    Signed 734: Boolean;
    ProgramName 733: String;
    PublisherLink 733: String;
    MoreInfoLink 732: String;
    Certificates 731: array of TCertificateInfo;
    Timestamped 736: Boolean;
    Timestamp 735: TDateTime;
    TimestampCertificates 735: array of TCertificateInfo;
end;
```

Remarks

This record contains information about digital signature (and its certificates) of executable file passed to SignatureGetInfo 558 function.

FileName 732 and Signed 734 fields are always valid. All other fields are valid only if Signed 734 field is `True`. Additionally, Timestamp 735 and TimestampCertificate fields are valid only if Timestamped 736 field is `True`.

See also

- SignatureGetInfo Function 558

11.87.1 Certificates

Signature's digital certificate.

Unit

ESignature 131

Syntax

Code (Delphi)

```
Certificates: array of TCertificateInfo;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EurekaLog CATCH EVERY BUG - EVERY TIME

Value
Information about digital certificate which was used to create digital signature.

Remarks
This field contains information about digital certificate which was used to create digital signature.

This field is valid only if Signed 734 field is `True`.

See also
- TimestampCertificate Field
- Signed Field 734
- TSignatureInfo Record 731

11.87.2 FileName

Original file name of executable.

Unit
ESignature 131

Syntax

Code (Delphi)

```
FileName: String;
```

Value
Executable file name.

Remarks
This is the same value as `AFileName` argument passed to SignatureGetInfo 558. This field is always valid.

Examples
```
'C:\Projects\MyApp.exe'
```

See also
- SignatureGetInfo Function 558
- TSignatureInfo Record 731

11.87.3 MoreInfoLink

Additional link.

Unit
ESignature 131

Syntax

Code (Delphi)

```
MoreInfoLink: String;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
Human-readable link.

Remarks
This field contains additional link.

This field may be empty, but it is often specified.

This field is valid only if Signed 734 field is `True`.

Examples
```
'http://www.eurekalog.com/'
```

See also
- PublisherLink Field 733
- Signed Field 734
- TSignatureInfo Record 731

11.87.4 ProgramName

Program name.

Unit
ESignature 131

Syntax

Code (Delphi)

```
ProgramName: String;
```

Value
Human-readable name.

Remarks
This field contains name which was used to sign executable.

This field is valid only if Signed 734 field is `True`.

Examples
```
'EurekaLog Tools Pack Installer'
```

See also
- Signed Field 734
- TSignatureInfo Record 731

11.87.5 PublisherLink

Publisher link.

Unit
ESignature 131

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
PublisherLink: String;
```

Value

Human-readable link.

Remarks

This field contains link defined by publisher.

This field is usually empty.

This field is valid only if Signed 734 field is `True`.

Examples

```
'http://www.eurekalog.com/'
```

See also

- MoreInfoLink Field 732
- Signed Field 734
- TSignatureInfo Record 731

11.87.6 Signed

Indicate if file has digital signature.

Unit

ESignature 131

Syntax

```
Code (Delphi)
Signed: Boolean;
```

Value

`True` - executable file is signed (has digital signature); `False` - file is not signed.

Remarks

This field is always valid.

All other fields (except FileName 732) are valid only if this field is `True`. If Signed is `False` then other fields (except FileName 732) are empty.

See also

- FileName Field 732
- Timestamped Field 736
- TSignatureInfo Record 731

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.87.7 Timestamp

Timestamp.

Unit
ESignature 131

Syntax

Code (Delphi)

```
Timestamp: TDateTime;
```

Value
Date and time of timestamp.

Remarks
This field contains date and time when executable was signed. The value must lie between `NotBefore` and `NotAfter` fields of Certificate field.

This field is valid only if Timestamped 736 field is `True`.

Examples
```
2015.10.23 14:48:02
```

See also
- TimestampCertificate Field
- TSignatureInfo Record 731

11.87.8 TimestampCertificates

Certificate used to sign timestamp.

Unit
ESignature 131

Syntax

Code (Delphi)

```
TimestampCertificates: array of TCertificateInfo;
```

Value
Information about certificate which was used to counter-sign the timestamp value.

Remarks
This field stores certificate which was used to counter-sign timestamp value. This certificate is different from certificate for digital signature (stored in Certificate field). This is certificate of timestamping service and it does not belong to the publisher (developer) of executable file.

The timestamp itself (date/time) is stored in Timestamp 735 field.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- Timestamped Field 736
- Timestamp Field 735
- Certificate Field
- TSignatureInfo Record 731

11.87.9 Timestamped

Indicate if signature is timestamped.

Unit

ESignature 131

Syntax

Code (Delphi)

```
Timestamped: Boolean;
```

Value

`True` - digital signature was timestamped; `False` - digital signature has no timestamp.

Remarks

This field is valid only if Signed 734 field is `True`.

If this field is `True`, then Timestamp 735 field and TimestampCertificate are also valid. Otherwise (`Timestamped` is `False`) Timestamp 735 and TimestampCertificate fields are zero.

See also

- Signed Field 734
- Timestamp Field 735
- TimestampCertificate Field
- SignatureGetInfo Function 558
- TSignatureInfo Record 731

11.88 TStreamConstructor

Function to Create a Compression/Decompression stream

Unit

EZip 144

Syntax

Code (Delphi)

```
TStreamConstructor = function(
   InStream: Classes.TStream;
   const ZipFile: TZipFile 266;
   const Item: TZipHeader 747;
   const AOwnStream: Boolean
): Classes.TStream;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.89 TSysErrorCode

A type for GetLastError/SetLastError.

Unit
ETypes 138

Syntax

Code (Delphi)

```
TSysErrorCode = Cardinal;
```

Remarks
EurekaLog introduces this type, because SysErrorMessage work with `Integer` in Delphi 4-2007, but works with `Cardinal` in Delphi 2009+. `GetLastError/SetLastError` work with `Cardinal` in all Delphi versions.

`TSysErrorCode` defines data type to be used with `GetLastError/SetLastError/SysErrorMessage`. Fixed `SysErrorMessage` routine is available in ECompatibility 62 unit.

11.90 TTEAKey

Key for TEA encryption.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
TTEAKey = packed array[0..3]of Cardinal;
```

Remarks
This type holds binary key which can be used for symmetrical encryption and decryption using TEA cipher.

Important!

This type is also used as salt storage. See PSalt 609/TSalt 721 type.

You do not need to free/release/dispose this key - it does not contain any allocated memory. However, it is strongly recommended to call TEADestroyKey 572 function once you have finished working with key - to erase it with zeros.

This key is derived from string (or binary) password via TEADeriveKey 565 function. You can freely export/import this key as RAW binary value, but you need to be careful - because the key have the same power as password. I.e. leaking this key means leaking your password.

Use TEAEncrypt 572 function to encrypt "clear text" data, use TEADecrypt 559 function to decrypt encrypted data and get "clear text" data back.

See also
- TEADeriveKey Function 565

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- TEAEncrypt Function 572
- TEADecrypt Function 559
- PSalt Type 609
- TSalt Type 721

11.91 TThreadKind

Indicates thread kind.

Unit

EThreadsManager 136

Syntax

Code (Delphi)

```
TThreadKind = (
    tkUnknown,
    tkGeneric 739,
    tkBeginThread 738,
    tkBeginThreadEx 739,
    tkTThread 739,
    tkTThreadEx 739
);
```

Members

tkBeginThread 738	RTL thread (created with BeginThread)
tkBeginThreadEx 739	RTL thread (created with BeginThreadEx function)
tkGeneric 739	Windows thread (created with CreateThread)
tkTThread 739	RTL thread (created with TThread's descendant)
tkTThreadEx 739	RTL thread (created with TThreadEx class)

Remarks

This type is used to specify thread kind in TEurekaThreadData class (see Kind property).

See also

- TEurekaThreadData Class 219
- TEurekaThreadData.Kind Property 224

11.91.1 tkBeginThread

RTL thread (created with BeginThread)

Unit

EThreadsManager 136

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TThreadKind Class 738

11.91.2 tkBeginThreadEx

RTL thread (created with BeginThreadEx function)

Unit

EThreadsManager 136

See also

- TThreadKind Class 738

11.91.3 tkGeneric

Windows thread (created with CreateThread)

Unit

EThreadsManager 136

See also

- TThreadKind Class 738

11.91.4 tkTThread

RTL thread (created with TThread's descendant)

Unit

EThreadsManager 136

See also

- TThreadKind Class 738

11.91.5 tkTThreadEx

RTL thread (created with TThreadEx class)

Unit

EThreadsManager 136

See also

- TThreadKind Class 738

11.92 TTwofishInitVector

Initialization Vector (IV) for Twofish.

Unit

EEncrypt 80

Syntax

Code (Delphi)

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

```
TTwofishInitVector = packed array[0..15]of Byte;
```

Remarks

Use TwofishInitIV 585 to obtain new random initialization vector. Pass the obtained value to TwofishEncrypt 583 and store it with encrypted data - to be passed into TwofishDecrypt 578 later.

Note
Initialization vector (IV) is a fixed-size input to a cryptographic primitive that is typically required to be random or pseudorandom. Randomization is crucial for encryption schemes to achieve semantic security, a property whereby repeated usage of the scheme under the same key does not allow an attacker to infer relationships between segments of the encrypted message.

See also

- TwofishEncrypt Function 583
- TwofishDecrypt Function 578
- TwofishInitIV Function 585

11.93 TTwofishKey

(Precalculated) Key for Twofish encryption.

Unit

EEncrypt 80

Syntax

Code (Delphi)

```
TTwofishKey = TEncryptBuffer 671;
```

Remarks

This is precalculated key used for symmetrical encryption and decryption by using Twofish cipher.

You need to free this key after you have done working with it - by using TwofishDestroyKey 583 function.

This key can be created by TwofishDeriveKey 580 function or be imported by TwofishInitKey 586 function. You need to be careful when handling this key. I.e. leaking this key means leaking your key/password.

Use TwofishEncrypt 583 function to encrypt "clear text" data, use TwofishDecrypt 578 function to decrypt encrypted data and get "clear text" back.

See also

- TTwofishRAWKey Type 740
- TwofishDeriveKey Function 580
- TwofishInitKey Function 586
- TwofishEncrypt Function 583
- TwofishDecrypt Function 578
- TwofishDestroyKey Function 583

11.94 TTwofishRAWKey

(RAW) Key for Twofish encryption.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
EEncrypt 80

Syntax

Code (Delphi)

```
TTwofishRAWKey = packed array[0..31]of Byte;
```

Remarks
This is RAW binary key for Twofish. 256 bits (32 bytes).

EurekaLog uses precalculated/cached key to speed up encryption/decryption. Therefore, RAW key can not be used directly. You must first "import" it via TwofishInitKey 586 function.

See also
- TTwofishKey Type 740
- TwofishInitKey Function 586

11.95 TZipCompression

Zip compression method.

Unit
EZip 144

Syntax

Code (Delphi)

```
TZipCompression = (
    zcStored 746,
    zcShrunk 746,
    zcReduce1 744,
    zcReduce2 744,
    zcReduce3 745,
    zcReduce4 745,
    zcImplode 743,
    zcTokenize 746,
    zcDeflate 743,
    zcDeflate64 743,
    zcPKImplode 744,
    zcReserved11 745,
    zcBZIP2 743,
    zcReserved13 745,
    zcLZMA 744,
    zcReserved15 745,
    zcReserved16 746,
    zcReserved17 746,
    zcTERSE 746,
    zcLZ77 743,
    zcWavePack 747 = 97,
    zcPPMdI1 744
);
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Members

zcBZIP2 743	File is compressed using BZIP2 algorithm
zcDeflate 743	The file is Deflated
zcDeflate64 743	Enhanced Deflating using Deflate64
zcImplode 743	The file is Imploded
zcLZ77 743	IBM LZ77 z Architecture (PFS)
zcLZMA 744	LZMA (EFS)
zcPKImplode 744	PKWARE Data Compression Library Imploding (old IBM TERSE)
zcPPMdI1 744	PPMd version I, Rev 1
zcReduce1 744	The file is Reduced with compression factor 1
zcReduce2 744	The file is Reduced with compression factor 2
zcReduce3 745	The file is Reduced with compression factor 3
zcReduce4 745	The file is Reduced with compression factor 4
zcReserved11 745	Reserved by PKWARE
zcReserved13 745	Reserved by PKWARE
zcReserved15 745	Reserved by PKWARE
zcReserved16 746	Reserved by PKWARE
zcReserved17 746	Reserved by PKWARE
zcShrunk 746	The file is Shrunk
zcStored 746	The file is stored (no compression)
zcTERSE 746	File is compressed using IBM TERSE (new)
zcTokenize 746	Reserved for Tokenizing compression algorithm

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

zcWavePack 747	WavPack compressed data

See also
- TZipCompressionToString function 586
- TZipFile class 266

11.95.1 zcBZIP2

File is compressed using BZIP2 algorithm

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.2 zcDeflate

The file is Deflated

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.3 zcDeflate64

Enhanced Deflating using Deflate64

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.4 zcImplode

The file is Imploded

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.5 zcLZ77

IBM LZ77 z Architecture (PFS)

Unit
EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TZipCompression Class 741

11.95.6 zcLZMA

LZMA (EFS)

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.7 zcPKImplode

PKWARE Data Compression Library Imploding (old IBM TERSE)

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.8 zcPPMdI1

PPMd version I, Rev 1

Unit

EZip 144

Requirements

IDE	Delphi 6 or above; C++ Builder 6 or above

See also

- TZipCompression Class 741

11.95.9 zcReduce1

The file is Reduced with compression factor 1

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.10 zcReduce2

The file is Reduced with compression factor 2

Unit

EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also

- TZipCompression Class 741

11.95.11 zcReduce3

The file is Reduced with compression factor 3

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.12 zcReduce4

The file is Reduced with compression factor 4

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.13 zcReserved11

Reserved by PKWARE

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.14 zcReserved13

Reserved by PKWARE

Unit

EZip 144

See also

- TZipCompression Class 741

11.95.15 zcReserved15

Reserved by PKWARE

Unit

EZip 144

See also

- TZipCompression Class 741

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

11.95.16 zcReserved16

Reserved by PKWARE

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.17 zcReserved17

Reserved by PKWARE

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.18 zcShrunk

The file is Shrunk

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.19 zcStored

The file is stored (no compression)

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.20 zcTERSE

File is compressed using IBM TERSE (new)

Unit
EZip 144

See also
- TZipCompression Class 741

11.95.21 zcTokenize

Reserved for Tokenizing compression algorithm

Unit
EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

See also
- TZipCompression Class 741

11.95.2.2 zcWavePack

WavPack compressed data

Unit
EZip 144

Requirements

IDE	Delphi 6 or above; C++ Builder 6 or above

See also
- TZipCompression Class 741

11.96 TZipEndOfCentralHeader

Final block written to zip file

Unit
EZip 144

Syntax

Code (Delphi)

```
TZipEndOfCentralHeader = packed record
   DiskNumber: UInt16;
   CentralDirStartDisk: UInt16;
   NumEntriesThisDisk: UInt16;
   CentralDirEntries: UInt16;
   CentralDirSize: UInt32;
   CentralDirOffset: UInt32;
   CommentLength: UInt16;
end;
```

Remarks
"Comments" field follows this record.

11.97 TZipHeader

TZipHeader contains information about a file in a zip archive.

Unit
EZip 144

Syntax

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
TZipHeader = packed record
    MadeByVersion 749: UInt8;
    HostOS 748: UInt8;
    RequiredVersion 749: UInt16;
    Flag: UInt16;
    CompressionMethod: UInt16;
    ModifiedDateTime: UInt32;
    CRC32: UInt32;
    CompressedSize: UInt32;
    UncompressedSize: UInt32;
    FileNameLength: UInt16;
    ExtraFieldLength: UInt16;
    FileCommentLength: UInt16;
    DiskNumberStart: UInt16;
    InternalAttributes: UInt16;
    ExternalAttributes: UInt32;
    LocalHeaderOffset: UInt32;
    FileName: String;
    FileNameRAW: RawByteString;
    ExtraField: TBytes;
    FileComment: String;
    FileCommentRAW: RawByteString;
    TimeCreate: TDateTime;
    TimeChange: TDateTime;
    TimeAccess: TDateTime;
end;
```

Remarks

This record is overloaded for use in reading/writing ZIP [Local file header] and the Central Directory's [file header].

See PKZIP Application Note section V. General Format of a .ZIP file sub section J. Explanation of fields for more detailed description of each field's usage.

11.97.1 HostOS

Version made by (HostOS byte; upper)

Unit

EZip 144

Syntax

Code (Delphi)

```
HostOS: UInt8;
```

Remarks

Indicates the compatibility of the file attribute information. If the external file attributes are compatible with MS-DOS and can be read by PKZIP for DOS version 2.04g then this value will be zero. If these attributes are not compatible, then this value will identify the host system on which the attributes are compatible. Software can use this information to determine the line record format for text files etc. The current mappings are:

- 0 - MS-DOS and OS/2 (FAT / VFAT / FAT32 file systems)
- 1 - Amiga

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- 2 - OpenVMS
- 3 - UNIX
- 4 - VM/CMS
- 5 - Atari ST
- 6 - OS/2 H.P.F.S.
- 7 - Macintosh
- 8 - Z-System
- 9 - CP/M
- 10 - Windows NTFS
- 11 - MVS (OS/390 - Z/OS)
- 12 - VSE
- 13 - Acorn Risc
- 14 - VFAT
- 15 - alternate MVS
- 16 - BeOS
- 17 - Tandem
- 18 - OS/400
- 19 - OS/X (Darwin)
- 20 thru 255 - unused

See also

- TZipHeader Record 747

11.97.2 MadeByVersion

Version made by (software byte; lower)

Unit

EZip 144

Syntax

Code (Delphi)

```
MadeByVersion: UInt8;
```

Remarks

Indicates the ZIP specification version (the version of this document) supported by the software used to encode the file. The value/10 indicates the major version number, and the value mod 10 is the minor version number.

Tip

Start of Central Header

See also

- TZipHeader Record 747

11.97.3 RequiredVersion

Version needed to extract (2 bytes)

Unit

EZip 144

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
RequiredVersion: UInt16;
```

Remarks

The minimum supported ZIP specification version needed to extract the file. This value is based on the specific format features a ZIP program must support to be able to extract the file. If multiple features are applied to a file, the minimum version should be set to the feature having the highest value. New features or feature changes affecting the published format specification will be implemented using higher version numbers than the last published value to avoid conflict.Current minimum feature versions are as defined below:

- 1.0 - Default value
- 1.1 - File is a volume label
- 2.0 - File is a folder (directory)
- 2.0 - File is compressed using Deflate compression
- 2.0 - File is encrypted using traditional PKWARE encryption
- 2.1 - File is compressed using Deflate64(tm)
- 2.5 - File is compressed using PKWARE DCL Implode
- 2.7 - File is a patch data set
- 4.5 - File uses ZIP64 format extensions
- 4.6 - File is compressed using BZIP2 compression (1)
- 5.0 - File is encrypted using DES
- 5.0 - File is encrypted using 3DES
- 5.0 - File is encrypted using original RC2 encryption
- 5.0 - File is encrypted using RC4 encryption
- 5.1 - File is encrypted using AES encryption
- 5.1 - File is encrypted using corrected RC2 encryption (2)
- 5.2 - File is encrypted using corrected RC2-64 encryption (2)
- 6.1 - File is encrypted using non-OAEP key wrapping (3)
- 6.2 - Central directory encryption
- 6.3 - File is compressed using LZMA
- 6.3 - File is compressed using PPMd (4)
- 6.3 - File is encrypted using Blowfish
- 6.3 - File is encrypted using Twofish

Note

(1) Early 7.x (pre-7.2) versions of PKZIP incorrectly set the version needed to extract for BZIP2 compression to be 50 when it should have been 46.

Note

(2) Refer to the section on Strong Encryption Specification for additional information regarding RC2 corrections.

Note

(3) Certificate encryption using non-OAEP key wrapping is the intended mode of operation for all versions beginning with 6.1. Support for OAEP key wrapping should only be used for backward compatibility when sending ZIP files to be opened by versions of PKZIP older than 6.1 (5.0 or 6.0).

Note

(4) Files compressed using PPMd should set the version needed to extract field to 6.3, however, not all ZIP programs enforce this and may be unable to decompress data files compressed using PPMd if this value is set. When using ZIP64 extensions, the corresponding value in the zip64 end of central directory record

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

should also be set. This field should be set appropriately to indicate whether Version 1 or Version 2 format is in use.

Tip
Start of Local Header

See also

- TZipHeader Record 747

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Part

12 All Globals

DefaultTracerMask 753	Mask for default call stack class.
EEurekaLogCurrentVersion 754	Current EurekaLog version (numeric).
EEurekaLogLicense 754	No description available
EEurekaLogVersion 755	Current EurekaLog version (textual).
EurekaCallStackClass 755	Default call stack class.
FreezeThreadClass 756	Determinate class to be used as background freeze detection thread.
LogBuilderClass 757	Specifies global log builder class to use.
CENTRALHEADER 757	Signature of .zip file.
DATADESCRIPTOR 758	Data descriptor signature.
LOCALHEADER 758	Local file header signature.
ZIPENDOFHEADER 759	End of central dir signature.

12.1 DefaultTracerMask

Mask for default call stack class.

Unit

ECallStack 59

Syntax

Code (Delphi)

```
DefaultTracerMask: Cardinal = TracerRawEurekaLogMask;
```

Value

Stack tracing mask to be used by default.

Remarks

This global variable holds mask for default call stack class. See EurekaCallStackClass 755 for more information about default class. Default class is used when you want to create stack tracing class to get call stack without knowing current preferences.

Important!

When you change this variable: be sure to change EurekaCallStackClass 755 as well.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Note
Default is mask for TEurekaStackListV7 218 class.

See also

- EurekaCallStackClass 755

12.2 EEurekaLogCurrentVersion

Current EurekaLog version (numeric).

Unit

EConsts 65

Syntax

```
Code (Delphi)
EEurekaLogCurrentVersion: DWord = 8000;
```

Value

Numeric value of EurekaLog's version.

Must be Word value for streaming purposes.

Example values: 6025, 6104, 7000, 7001, etc.

Remarks

This constant represents EurekaLog's version number in the following format: XYZZ = 'X.Y.ZZ'. For example, 6104 means '6.1.04'.

This value is used for streaming EurekaLog data formats. You can use EEurekaLogVersion 755 constant for human-readable version information.

Note
This value may be less than actual EurekaLog version.

See also

- EEurekaLogVersion 755

12.3 EEurekaLogLicense

EurekaLog's license type.

Unit

EConsts 65

Syntax

```
Code (Delphi)
EEurekaLogLicense = 'Enterprise';
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
String description of current EurekaLog license.

Example values: 'Trial', 'Professional', 'Enterprise'.

For non-stable releases a '(beta)' suffix is added. Example: 'Professional (beta)'.

Remarks
Each EurekaLog is compiled against specific source code - trial, professional, enterprise, etc. This constant contains value of license for which current source code was compiled.

Note
You can not change this value until you have a full source code of EurekaLog (to recompile EurekaLog). Currently source code is available only for Enterprise edition of EurekaLog. When you have source code, you can change this constant to arbitrary value.

See also
- EEurekaLogVersion 755

12.4 EEurekaLogVersion

Current EurekaLog version (textual).

Unit
EConsts 65

Syntax

Code (Delphi)
```
EEurekaLogVersion = '7.13.0.0';
```

Value
String representation of current EurekaLog's version.

Example value: '7.0.0.177'.

For beta versions, RC versions and other non-stable releases a suffix is added. Example: '7.0.0.177 RC 2'

Remarks
This constant represents EurekaLog's version number in the standard 4-number format 'A.B.C.D' plus optional suffix for non-stable releases.

See also
- GetEurekaLogVersion 418
- EEurekaLogCurrentVersion 754
- EEurekaLogLicense 754

12.5 EurekaCallStackClass

Default call stack class.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Unit
ECallStack 59

Syntax

Code (Delphi)
```
EurekaCallStackClass: TEurekaCallStackClass 678 = TEurekaStackList;
```

Value
Stack tracing class to be used by default.

Remarks
This global variable holds default call stack class. Default class is used when you want to create stack tracing class to get call stack without knowing current preferences.

Use this variable to create call stack objects. Use GetTracer 430/ GetTracerClass 431 to use specific classes.

You can replace it to use your own custom call stacks with all EurekaLog's routines (however, this is not recommended).

Important!

When you change this variable: be sure to change DefaultTracerMask 753 as well.

Note

Default is TEurekaStackListV7 218 class.

See also
- DefaultTracerMask 753
- GetTracer 430
- GetTracerClass 431

12.6 FreezeThreadClass

Determinate class to be used as background freeze detection thread.

Unit
EFreeze 96

Syntax

Code (Delphi)
```
FreezeThreadClass: TFreezeThreadClass 686;
```

Value
Class to create by InitCheckFreeze 436 function.

Remarks
This variable is used by InitCheckFreeze 436 function to create a new background freeze detection thread. You can use build-in classes (TMainThreadFreezeDetectionThread 254 or TWCTFreezeDetectionThread 265) or

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

implement your own freeze detection thread class. Class has to be descendant of TFreezeThread 246 and it must override Execute 248 method.

Note
You need to re-initialize freeze detection after changing this variable if freeze checks are running. Use DoneCheckFreeze 357 and InitCheckFreeze 436 to stop/start background freeze detection thread. You do not need to re-initialize freeze detection if you change this variable before initialization of EurekaLog. The best course of action would be disabling freeze detection in EurekaLog options, then assigning your own freeze detection class and enabling freeze detection via a call to `InitCheckFreeze` function.

See also

- TFreezeThread Class 246
- InitCheckFreeze Function 436
- DoneCheckFreeze Function 357
- TFreezeThreadClass Type 686
- TMainThreadFreezeDetectionThread Class 254
- TWCTFreezeDetectionThread Class 265

12.7 LogBuilderClass

Specifies global log builder class to use.

Unit

ELogBuilder 101

Syntax

Code (Delphi)

```
LogBuilderClass: TBaseLogBuilderClass = TLogBuilder;
```

Value

A specific log builder class. TLogBuilder 253 is the default value.

Remarks

This global variable specifies class which will be used to build reports by EurekaLog. Default is the TLogBuilder class. You can set this variable to your own class to alter content of all text bug reports.

12.8 CENTRALHEADER

Signature of .zip file.

Unit

EZip 144

Syntax

Code (Delphi)

```
SIGNATURE_CENTRALHEADER: UInt32 = $02014B50;
```

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Value
'PK'#1#2 (4 bytes)

Remarks
To be used as signature value in zip file header record.

See also
- TZipHeader 747

12.9 DATADESCRIPTOR

Data descriptor signature.

Unit
EZip 144

Syntax

Code (Delphi)

```
SIGNATURE_DATADESCRIPTOR: UInt32 = $08074B50;
```

Value
'PK'#7#8 (4 bytes)

Remarks
To be used as signature value for data descriptor record.

Although not originally assigned a signature, the value $08074b50 has commonly been adopted as a signature value for the data descriptor record. Implementers should be aware that ZIP files may be encountered with or without this signature marking data descriptors and should account for either case when reading ZIP files to ensure compatibility. When writing ZIP files, it is recommended to include the signature value marking the data descriptor record. When the signature is used, the fields currently defined for the data descriptor record will immediately follow the signature.

12.10 LOCALHEADER

Local file header signature.

Unit
EZip 144

Syntax

Code (Delphi)

```
SIGNATURE_LOCALHEADER: UInt32 = $04034B50;
```

Value
'PK'#3#4 (4 bytes)

Remarks
To be used as signature value in local file header record.

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

12.11 ZIPENDOFHEADER

End of central dir signature.

Unit

EZip 144

Syntax

Code (Delphi)

```
SIGNATURE_ZIPENDOFHEADER: UInt32 = $06054B50;
```

Value

'PK'#5#6 (4 bytes)

Remarks

To be used as signature value in end of central directory record 747.

See also

- TZipEndOfCentralHeader 747

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

Index

- A -

Accessible 687
AcquireThreadData 318
ActiveControl 183
ActiveControlClass 184
ActiveControlText 184
ActiveForm 184
ActiveFormClass 185
ActiveFormText 185
Add 269
AddCheckPoint 280
AddCustomDataToBugReport 319
AddCustomWebField 320
AdditionalInfo 186
AddResetSeparator 280
Address 186, 663
AddSeparator 280
AllocatePageAlign 321
AllocationGranularity 322
AllocMemCountEx 322
AllocMemCountMax 323
AllocMemSizeEx 324
AllocMemSizeMax 324
atCollectedExceptionInfo 674
atCollectingExceptionInfo 675
atEurekaLogInitialized 675
atEurekaLogInitializing 675
atEventHandler 675
atExceptionHandled 675
atExceptionHandling 676
atGeneral 676
atRestarting 676
atSavedLogFile 676
atSavingLogFile 676
atSendCancelled 676
atSending 677
atSent 677
atShowedExceptionInfo 677
atShowingExceptionInfo 677
atTerminating 677
AutoHandleException 263

- B -

BaseException 186
BeginThreadEx 325
BinaryFileName 663
BlockState 711
BlockType 711
btCppObject 612
btData 612
btDynArray 612
btObject 613
btString 613
BugID 187
BugIDSource 188
BugIDStr 188
Build 178
BuildBugReport 328
BuildReport 153

- C -

C 295
CallerAddress 221
CallerFunction 222
CallStack 156, 188, 712
CanCallMemWipeStack 328
Category 281
Certificates 731
CheckELMagicCode 329
CheckHeap 330
ChecksPaused 247
ClassName 663
ClearThreadExceptions 234
Close 271
Color 281
Comment 274
ConfigDelete 331
ConfigFileName 331
ConfigIsRegistry 333
ConfigKeyPath 334
ConfigReadBool 335
ConfigReadFloat 336
ConfigReadInteger 337
ConfigReadString 338
ConfigWriteBool 339
ConfigWriteFloat 340
ConfigWriteInteger 341
ConfigWriteString 342
Context 189
ConvertAddress 343
ConvertRetAddr 344
Count 230
CRC16Hash 345
CRC32Hash 346
Create 153, 176, 250, 258, 268

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

ctEqual 620
ctGetLastError 620
ctHResult 620
ctNone 621
ctNotEqual 621
ctNTStatus 621
CurrentEurekaLogOptions 8, 347
CurrentInstance 348
Cursor 189
CursorPos 189
CustomFields 157
CustomID 190

- D -

D 297
DataProtect 349
DataUnprotect 350
ddModule 679
ddNone 679
ddProcedure 679
ddSourceCode 680
ddUnit 680
DebugBreak 352
DebugDetail 664
DebugInfo 664
DecodeConfigValue 352
DecodeIdent 160
DecodeInstruction 353
DefaultTracerMask 753
Destroy 260, 269
Details 282
DisableMemLeaksRegistering 354
DisasmFunction 355
DisasmInstruction 356
DoneCheckFreeze 357
DumpAllocationsToFile 358

- E -

EAppCLX 53
EAppCrossModule 53
EAppFMX 54
EAppHTTPApp 54
EAppType 54
EAppVCL 56
EAppWebBroker 56
EBase 57
EBaseClasses 58
ECallStack 59
EClasses 61
EClassUtils 61
ECompatibility 62
EComponent 62
EConfig 63
EConsts 65
ECore 65
EDbgHelp 65
EDebugDWARF 66
EDebugEL 66
EDebugEL7 67
EDebugExportBase 67
EDebugExports 67
EDebugInfo 68
EDebugJCL 69
EDebugMAB 69
EDebugMAB2 69
EDebugMap 70
EDebugME 70
EDebugMS 70
EDebugStabs 71
EDebugTD32 71
EDialog 72
EDialogCGI 73
EDialogConsole 73
EDialogHTMLBased 74
EDialogSend 74
EDialogSendWinAPI 74
EDialogService 75
EDialogWeb 75
EDialogWER 75
EDialogWinAPI 76
EDialogWinAPIEurekaLog 76
EDialogWinAPIEurekaLogDetailed 76
EDialogWinAPIMSClassic 77
EDialogWinAPIStepsToReproduce 77
EDisAsm 78
EDLLs 78
EDNS 79
EDNSError 151
EDWARF 79
EEncoding 79
EEncrypt 80
EEurekaLogCurrentVersion 754
EEurekaLogLicense 754
EEurekaLogVersion 755
EEvents 83
EException 89
EExceptionHook 90
EExceptionInfo 90
EExceptionInfoBCB 91
EExceptionInfoCLR 91
EExceptionInfoCpp 91

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EExceptionInfoDelphi2 92
EExceptionInfoDelphiANSI 92
EExceptionInfoDelphiUnicode 92
EExceptionInfoGeneric 92
EExceptionInfoLLVM 93
EExceptionInfoMSVC 93
EExceptionManager 93
EFileMemory 95
EFixSafeCallException 95
EFonts 95
EFreeze 96
EHash 97
EHook 99
EHTMLHelp 99
EInfoFormat 99
EInject 100
EInterfaces 100
EJSON 100
EListView 100
elmcBitmap 695
elmcCollection 695
elmcColor 695
elmcComponentAsText 696
elmcCustomHierarchy 696
elmcCustomProperties 696
elmcCustomTable 696
elmcCustomText 696
elmcMemory 697
elmcMemoryStatus 697
elmcObject 697
elmcRegistry 697
elmcStackTrace 697
elmcStringList 697
elmcSystemInfo 698
elmcText 698
elmcTextFile 698
elmcVersionInfo 698
elmcWindow 698
elmcXmlData 699
elmtAppStartSeparator 699
elmtBlue 699
elmtCheckPoint 699
elmtClear 699
elmtCollapsedMethod 699
elmtEnterMethod 700
elmtError 700
elmtException 700
elmtExitMethod 700
elmtExitMethodCollapse 700
elmtGreen 701
elmtInactiveSeparator 701
elmtIndigo 701
elmtInfo 701
elmtLevel1 701
elmtLevel2 701
elmtLevel3 702
elmtLevel4 702
elmtLevel5 702
elmtLevel6 702
elmtLevel7 702
elmtNote 703
elmtOrange 703
elmtRed 703
elmtReminder 703
elmtResetSeparator 703
elmtSaveLogFile 703
elmtScratchPad 704
elmtSeparator 704
elmtTiming 704
elmtViolet 704
elmtWarning 704
elmtYellow 705
ELog 360
ELogAddCheckPoint 364
ELogAddResetSeparator 364
ELogAddSeparator 365
ELogAssigned 365
ELogAssignedPtr 366
ELogBuilder 101
ELogClear 367
ELogCurrency 367
ELogDateTime 368
ELogEnter 369
ELogError 371
ELogException 371
ELogFlush 372
ELogging 103
ELogIf 373
ELogIfDateTime 374
ELogInterval 375
ELogManager 105
ELogMemoryAsHex 376
ELogNote 377
ELogOpen 378
ELogPointer 379
ELogReminder 380
ELogResetCheckPoint 380
ELogWarning 381
ELogWindowHandle 382
ELogWinError 383
ELowLevel 105
ELowLevelClasses 108
ELowLevelExceptions 108
EMantisConnectStub 108

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EMAPI 108
EMapParserBase 109
EMapWin32 109
emD2009 685
EMemLeaks 109
emFull 686
emHook 686
EMLang 113
emNull 686
EModules 114
EMonitors 114
emUnknown 686
EnableMemLeaksRegistering 383
EncodeConfigValue 384
EncryptBufferFree 385
Encryption 614
EndThreadEx 386
EnsureThreadStack 387
ENT 114
Enter 283
EnumSenders 387
EOSApiList 115
EPChars 115
EPEImage 115
EPNG 115
EResLeaks 116
EResourceStrings 116
ErrorCode 713
ErrorMessage 714
ERunTime 116
EScreenShot 119
ESend 120
ESendAPIBugZilla 121
ESendAPIExceptionless 122
ESendAPIFogBugz 122
ESendAPIGitHub 123
ESendAPIGitLab 124
ESendAPIJIRA 124
ESendAPIJSON 125
ESendAPIMantis 125
ESendAPIRedmine 126
ESendAPIYouTrack 126
ESendMail 127
ESendMailMAPI 127
ESendMailShell 128
ESendMailSMAPI 128
ESendMailSMTP 129
ESendWeb 129
ESendWebFTP 130
ESendWebHTTP 130
ESignature 131
ESOAP 132
ESOAPMantis 132
ESocket 132
ESockets 132
ESpecificDelphi 133
ESSLSocket 133
EStackTracing 133
EStrConsts 134
ESysInfo 134
ETaskDialogs 136
EThreadsManager 136
ETools 137
ETypes 138
EUnmangling 139
eupChild 671
eupNone 671
eupParent 671
EurekaAllocMem 388
EurekaCallStackClass 755
EurekaFreeMem 389
EurekaGetMem 390
EurekaReallocMem 391
EurekaRegisterExpectedMemoryLeak 392
EurekaRegisterExpectedMemoryLeakDynArray 393
EurekaRegisterExpectedMemoryLeakStr 393
EurekaTryGetMem 394
EurekaUnregisterExpectedMemoryLeak 395
EurekaUnregisterExpectedMemoryLeakDynArray 395
EurekaUnregisterExpectedMemoryLeakStr 396
EWCT 139
EWCTSupport 140
EWebTools 140
EWER 140
EWinCrypt 141
EWinSock 141
ExceptionClass 190
ExceptionCode 190
ExceptionCount 244
ExceptionExternal 191
ExceptionInfo 157
ExceptionLocation 191
ExceptionLog7 141
ExceptionManager 396
ExceptionMessage 192, 397
ExceptionNative 192
ExceptionNumber 193
ExceptionObject 193
ExceptionRecord 195
ExceptionThreadID 195
Execute 248, 261
EXML 143

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

EXMLBuilder 143
EXMLRPC 143
EXMLRPCBugZilla 144
ExpandEnvVars 398
Expected 195
ExpectedContext 196
ExpectedURL 197
Extract 271
ExtractAll 272
ExtractZipFile 267
EZip 144
EZIPException 151
EZLib 145
EZLibHeaders 145

- F -

FBuilded 156
FileComment 275
FileCount 275
FileInfo 275
FileInfos 275
FileName 276, 732
FileNames 276
FindDialogClass 399
FindLowAddress 400
FindScreenshotRect 400
FindSender 401
FontColor 283
FreezeNotifyStillAlive 401
FreezeThreadClass 756
FreezeTimeout 248
ftBugReport 682
ftHTMLPage 682
ftLog 682
ftMiniDump 682
ftSavedBugReport 682
ftScreenshot 683
ftXML 683
ftZIP 683
ftZIPCopy 683

- G -

GetActiveFormClass 402
GetActiveFormText 403
GetApplicationDescription 404
GetApplicationName 404
GetApplicationRect 405
GetApplicationWnd 406
GetAsmSize 406
GetCompanyName 407
GetCompilationDate 408
GetComputerName 409
GetControlClass 409
GetControlText 410
GetCurrentInstruction 173
GetCurrentModuleCompilationDate 411
GetCurrentStackFrame 174
GetCurrentStackPointer 175
GetCurrentTopOfStack 175
GetCurrentUserPrivileges 411
GetDiskInfo 412
GetDisplayDPI 412
GetDisplayMode 413
GetEnvVariable 413
GetEurekaLogLabel 414
GetEurekaLogLicense 415
GetEurekaLogLicenseType 416
GetEurekaLogPath 417
GetEurekaLogStatus 417
GetEurekaLogVersion 418
GetExceptionModuleName 419
GetExceptionModuleVersion 419
GetFileIcon 420
GetFileSize 421
GetFunctionSize 422
GetLocationInfo 423
GetLocationInfoStr 423
GetMainWnd 424
GetModuleType 425
GetMXServerFromEmail 426
GetParentWnd 426
GetPerformanceCount 427
GetReadableSize 428
GetTaskBarWnd 429
GetTracer 430
GetTracerClass 431
GetUserEMail 432
GUIFont 432

- H -

Handle 222, 234
Handled 197
HandleException 433
Handler 198
Hash 614
HookWin32API 434
HostOS 748

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

- I -

ID 223
IEurekaLogger 278
IEurekaLoggerParams 295
IfElse 435
IndexOf 272
Info 198, 237
IniReadString 435
InitCheckFreeze 436
InitializationException 199
InitSalt 438
InitWER 440
IniWriteString 440
InnerException 199
Instance 233
InstructionToString 441
IntoIDE 442
IsAppType 442
IsBPL 443
IsCBuilder 444
IsCGI 445
IsCGIConsole 445
IsCGIWin 446
IsCLX 446
IsCompiledWithPackages 447
IsConsole 447
IsCPL 448
IsDebugged 449
IsDLL 449
IsDynamicRTL 450
IsEurekaLogActive 3, 450, 452
IsEurekaLogActiveInThread 5, 453
IsEurekaLogInstalled 2, 454
IsEXE 455
IsExpected 455
IsFMX 456
IsFreezeCheckEnabled 457
IsFreezeCheckInstalled 458
IsGUI 458
IsIntraWeb 459
IsISAPI 459
IsLLVM 460
IsMainThread 223
IsMemLeaksEnabled 461
IsMemLeaksInstalled 461
IsModuleLibrary 462
IsReadableFile 463
IsRTLCompiledWithStackFrames 463
IsRTLThread 223
IsService 464
IsStackFramesProc 464
IssueID 714
IssueIDStr 715
IssueIDType 715
Issuer 615
IssuerID 615
IsValid 267
IsValidBlockAddr 465
IsValidClass 466
IsValidCodeAddr 467
IsValidHandle 468
IsValidObject 468
IsVCL 469
IsVirtualized 470
IsWeb 470
IsWin10 471
IsWin11 472
IsWin64 472
IsWin7 473
IsWin8 474
IsWin81 474
IsWindowsThread 224
IsWinVista 475
IsWinXP 476
IsWorkstation 476
IsWritableFile 477
Items 231

- K -

Kind 224, 284

- L -

LastBugID 9, 478
LastSystemErrorCode 200
LastThreadException 244
leHang 706
leInvalidBlock 707
leMultiFree 707
leNone 707
leObjectInUse 707
leOutOfMemory 707
leOverflow 708
leUninstalled 708
leUseAfterFree 708
LineNumber 665
LocationToStr 478
Log 285
LogAssigned 287

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

LogAssignedPtr 287
LogBuilderClass 757
LogCurrency 288
LogDateTime 288
LogError 288
LogException 289
LogIf 289
LogIfDateTime 290
LogInterval 291
LogMemoryAsHex 291
LogNote 291
LogPointer 292
LogReminder 292
LogWarning 293
LogWindowHandle 293
LogWinError 294

- M -

MadeByVersion 749
MainThreadTopOfStack 479
MapStringToFileName 160
MapStringToModuleName 161
MapStringToStr 162
MarkExpected 480
MD5Hash 480
MD5HMAC 481
MemLeaksGetName 482
MemLeaksName 483
MemLeaksNameDynArray 484
MemLeaksNameStr 485
MemLeaksOwn 485
MemLeaksOwnDynArrayBoth 486
MemLeaksOwnDynArrayChild 487
MemLeaksOwnDynArrayParent 487
MemLeaksOwnStr 488
MemLeaksSetParentBlock 489
MemLeaksSetParentBlockDynArrayBoth 489
MemLeaksSetParentBlockDynArrayChild 489
MemLeaksSetParentBlockDynArrayParent 490
MemLeaksSetParentBlockStr 491
MemProtect 491
MemUnprotect 493
MemWipeStack 494
Mode 245, 276
Module 200, 665
ModuleName 200, 666
ModuleType 666
ModuleVersion 201
MonoFont 495
MoreInfoLink 732
mtLibrary 709
mtPackage 710
mtProgram 710
mtUnknown 710

- N -

Name 225
NameThread 496
NameThreadForDebugging 496
Next 688
NotAfter 616
NotBefore 617
Notify 239

- O -

Offset 666
OffsetFromLineNumber 667
OffsetFromProcName 667
OnAttachedFilesRequest 210
OnBeginReportGen 211
OnCustomBugID 211
OnCustomButtonClick 212
OnCustomDataRequest 212
OnCustomFileName 212
OnCustomWebFieldsRequest 213
OnEndReportGen 213
OnExceptionAction 214
OnExceptionError 214
OnExceptionNotify 214
OnPasswordRequest 215
OnRaise 215
OnReproduceRequest 215
OnZippedFilesRequest 216
Open 273
Options 158, 201

- P -

P 298
PageSize 497
Parameter 225
ParentID 226
PauseFreezeCheck 497
PCustomName 607
PELLocationInfo 608
PInstruction 608
PlatformSpecific 688
PPointerRecord 608
PrepareStr 163

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

PrimaryInstance 498
Priority 220
PrivateKey 719
ProcedureName 668
Processed 202
ProcOffsetLine 668
ProgramName 733
PSalt 609
PtrUInt 609
PTwofishInitVector 610
PublicKey 720
PublisherLink 733

- R -

RaiseExpected 499
RaiseFreezeException 499
RaiseOuterException 500
RaiseOuterExceptionExpected 501
Read 273
Reader 202
RealParameter 226
RealStartingAddress 226
RealStartingFunction 227
Refresh 154
RegisterDialogClass 502
RegisterDialogClassFirst 503
RegisterEventAttachedFilesRequest 504
RegisterEventBeginReportGen 505
RegisterEventCustomBugID 506
RegisterEventCustomButtonClick 507
RegisterEventCustomDataRequest 508
RegisterEventCustomFileName 509
RegisterEventCustomWebFieldsRequest 510
RegisterEventEndReportGen 511
RegisterEventExceptionAction 512
RegisterEventExceptionError 513
RegisterEventExceptionNotify 514
RegisterEventHelpButtonClick 515
RegisterEventPasswordRequest 516
RegisterEventRaise 517
RegisterEventReproduceRequest 518
RegisterEventZippedFilesRequest 519
RegisterSender 520
RegisterSenderFirst 520
RegKeyCheck 521
RegKeyRead 522
RegKeyWrite 523
Report 158
ReproduceText 203
RequiredVersion 749
ResetCheckPoint 294
ResumeFreezeCheck 524
Ret 300
RSADecrypt 524
RSADestroyKey 527
RSAEncrypt 528
RSAGenKey 531
RSALoadPrivateKey 532
RSALoadPublicKey 533
RSASavePrivateKey 534
RSASavePublicKey 535
RSASign 536
RSAVerify 538
rsBLOB 718
rsDER 718
rsPEM 718

- S -

SafeCallExpt 203
SafeExec 540
Screenshot 541
ScreenshotFree 543
ScreenshotInit 543
ScreenshotSave 544
SDBMHash 545
Secondary 204
SecureFree 545, 547
SendResult 716
Serial 617
SetAppType 547
SetEurekaLogState 6, 548, 549
SetEurekaLogStateInThread 7, 550
SetExceptionMessage 551
SetupConfig 552
SetUserEmail 553
SHA1Hash 553
SHA1HMAC 554
SHA256Hash 555
SHA256HMAC 556
ShowExpected 557
ShowLastExceptionData 240
SIGNATURE_CENTRALHEADER 757
SIGNATURE_DATADESCRIPTOR 758
SIGNATURE_LOCALHEADER 758
SIGNATURE_ZIPENDOFHEADER 759
SignatureGetInfo 558
Signed 734
SingleSize 712
SourceName 669
srBugClosed 724

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

srBugReopen 725
srCancelled 725
srConnectionError 725
srInvalidInsert 725
srInvalidLogin 725
srInvalidModify 726
srInvalidRead 726
srInvalidSearch 726
srInvalidSelection 726
srNoExceptionInfo 726
srObsoleteVersion 726
srRestart 727
srSent 727
srUnknownError 727
srUnsupportedSender 727
ssActiveWindow 729
ssApplication 729
ssDesktop 729
ssForegroundWindow 729
ssNone 729
ssPrimary 730
ssWorkarea 730
StackDump 204
StackPoint 205
StackTop 205, 227
StandardEurekaError 241
StandardEurekaNotify 242
Start 689
StartingAddress 228
StartingFunction 228
Subject 618
SubjectID 618
SuccessMessage 716
SynchronizeException 206

- T -

TAppType 610
TargetThreadHandle 249
TargetThreadID 249
TargetWindow 249
TBaseDialog 36, 151
TBaseDialogClass 611
TBaseLogBuilder 152
TBlockType 611
TBugID 613
TCertificateInfo 613
TCheckType 619
TCRC16Hash 621
TCRC32Hash 622
TCustomName 622
TDummyLogBuilder 159
TDumpCompareFunc 622
TDumpFilterFunc 623
TEADecrypt 559
TEADefaultKey 563
TEADeriveKey 565
TEADeriveKeyHMAC 569
TEADestroyKey 572
TEAEncrypt 572
TEAInitSessionKey 576
TELAbstractMapParser 159
TELDebugInfoSource 164
TELEvAttachedFilesRequestMeth 21, 624
TELEvAttachedFilesRequestProc 625
TELEvCustomBugIDMeth 25, 625
TELEvCustomBugIDProc 626
TELEvCustomButtonClickMeth 28, 627
TELEvCustomButtonClickProc 628
TELEvCustomDataRequestMeth 26, 629
TELEvCustomDataRequestProc 631
TELEvCustomFileNameMeth 24, 632
TELEvCustomFileNameProc 633
TELEvCustomWebFieldsRequestMeth 27, 634
TELEvCustomWebFieldsRequestProc 635
TELEvExceptionActionMeth 16, 636
TELEvExceptionActionProc 638
TELEvExceptionErrorMeth 18, 640
TELEvExceptionErrorProc 642
TELEvExceptionNotifyMeth 14, 643
TELEvExceptionNotifyProc 645
TELEvHelpButtonClickMeth 29, 647
TELEvHelpButtonClickProc 648
TELEvPasswordRequestMeth 19, 649
TELEvPasswordRequestProc 651
TELEvRaiseMeth 12, 652
TELEvRaiseProc 653
TELEvReportGenMeth 30, 655, 656
TELEvReportGenProc 657
TELEvReproduceRequestMeth 23, 658
TELEvReproduceRequestProc 659
TELEvZippedFilesRequestMeth 22, 660
TELEvZippedFilesRequestProc 661
TELFTPSender 43, 165
TELHTTPSender 43, 165
TELLocationInfo 662
TELMailMAPISender 42, 165
TELMailShellSender 41, 166
TELMailSimpleMAPISender 41, 166
TELMailSMTPBaseSender 166
TELMailSMTPClientSender 43, 167
TELMailSMTPSender 42, 167
TELMailSMTPServerSender 42, 167

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

TELTrackerBugZillaSender 168
TELTrackerExceptionlessSender 168
TELTrackerFogBugzSender 45, 169
TELTrackerGitHubSender 169
TELTrackerGitLabSender 170
TELTrackerJIRASender 170
TELTrackerMantisSender 44, 170
TELTrackerRedmineSender 171
TELUniversalSender 171
TELUniversalSenderClass 669
TELUseParent 670
TEncryptBuffer 671
TEurekaActionType 672
TEurekaBaseStackList 172
TEurekaCallStack 678
TEurekaCallStackClass 678
TEurekaDebugDetail 678
TEurekaDebugDetails 680
TEurekaExceptionInfo 180
TEurekaLogEvents 33, 208
TEurekaLogFileType 680
TEurekaModuleOptions 216
TEurekaModuleType 683
TEurekaStackList 684
TEurekaStackListV6 217
TEurekaStackListV7 218
TEurekaThreadData 219
TEurekaThreadDataList 229
TExceptionManager 232
TExceptionNotifyMode 684
TFreezeThread 246
TFreezeThreadClass 686
TGenericStackList 249
Thread 229
ThreadClassName 229
ThreadData 231, 577
ThreadDataEnum 578
ThreadException 206
ThreadID 207
Time 208
Timestamp 735
TimestampCertificates 735
Timestamped 736
TInstruction 687
TIssueIDType 690
TJSONLogBuilder 252
tkBeginThread 738
tkBeginThreadEx 739
tkGeneric 739
tkTThread 739
tkTThreadEx 739
TLogBuilder 253
TLogMsgContent 691
TLogMsgType 691
TLogType 692
TMainThreadFreezeDetectionThread 254
TMD5Hash 705
TMemoryError 705
TMemoryErrors 708
TModuleType 709
ToString 155, 180
TPointerRecord 710
Tracer 252
TResponse 713
TRSAExport 717
TRSAImport 718
TRSAKey 719
TRSASignature 720
TSalt 721
TScreeenshotHandle 721
TSDBMHash 722
TSendResult 722
TSendScreenshot 727
TSHA1Hash 730
TSHA256Hash 730
TSignatureInfo 731
TStreamConstructor 736
TSysErrorCode 737
TTEAKey 737
TThreadEx 255
TThreadKind 738
TTwofishInitVector 739
TTwofishKey 740
TTwofishRAWKey 740
TWCTFreezeDetectionThread 265
TwofishDecrypt 578
TwofishDeriveKey 580
TwofishDeriveKeyHMAC 582
TwofishDestroyKey 583
TwofishEncrypt 583
TwofishInitIV 585
TwofishInitKey 586
TXMLLogBuilder 265
TZipCompression 741
TZipCompressionToString 586
TZipEndOfCentralHeader 747
TZipFile 266
TZipHeader 747

- U -

UnhandledExceptionCount 245
UnhandledExceptionNumber 208

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.

UnitName 669
UnRegisterEventAttachedFilesRequest 587
UnRegisterEventBeginReportGen 588
UnRegisterEventCustomBugID 589
UnRegisterEventCustomButtonClick 589
UnRegisterEventCustomDataRequest 590
UnRegisterEventCustomFileName 591
UnRegisterEventCustomWebFieldsRequest 592
UnRegisterEventEndReportGen 593
UnRegisterEventExceptionAction 594
UnRegisterEventExceptionError 594
UnRegisterEventExceptionNotify 595
UnRegisterEventHelpButtonClick 596
UnRegisterEventPasswordRequest 597
UnRegisterEventRaise 598
UnRegisterEventReproduceRequest 598
UnRegisterEventZippedFilesRequest 599

- V -

Valid 689
Version 619

- W -

WCTDone 600
WCTInit 600

- Z -

zcBZIP2 743
zcDeflate 743
zcDeflate64 743
zcImplode 743
zcLZ77 743
zcLZMA 744
zcPKImplode 744
zcPPMdI1 744
zcReduce1 744
zcReduce2 744
zcReduce3 745
zcReduce4 745
zcReserved11 745
zcReserved13 745
zcReserved15 745
zcReserved16 746
zcReserved17 746
zcShrunk 746
zcStored 746
zcTERSE 746
zcTokenize 746
zcWavePack 747
ZipDirectoryContents 268

© 2001-2018, Neos Eureka S.r.l. All Rights Reserved.