

EurekaLog 7.x.x code reference

© 2001-2015, 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 ... 15
4 OnExceptionError ... 17
5 OnPasswordRequest ... 18
6 OnAttachedFilesRequest ... 19
7 OnZippedFilesRequest ... 20
8 OnReproduceRequest ... 21
9 OnCustomFileName ... 21
10 OnCustomBugID ... 22
11 OnCustomDataRequest ... 23
12 OnCustomWebFieldsRequest ... 23
13 OnCustomButtonClick ... 24
14 OnHelpButtonClick ... 25
15 OnBeginReportGen/OnEndReportGen ... 26

Part III Component 29

1 Events ... 30

Part IV Dialogs 32

1 Base class ... 32
2 Null dialog ... 33
3 RTL dialog ... 33
4 Console ... 33
5 MessageBox dialog ... 33
MessageBox detailed dialog ... 33
6 MS Classic ... 33
7 EurekaLog ... 33
EurekaLog detailed ... 33

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8 CGI ... 33
9 ISAPI ... 33
10 IntraWeb ... 33
11 WER ... 33
12 System Log ... 33

Part V Debug information providers 35

1 EurekaLog ... 35
2 .map files ... 35
3 Turbo Debugger (TDS) ... 35
4 JEDI (Jcl) ... 35
5 madExcept ... 35
6 Microsoft ... 35
7 DLL export table ... 35

Part VI Send engines 37

1 mailto ... 37
2 Simple MAPI ... 37
3 MAPI ... 38
4 SMTP ... 38
SMTP server ... 38
SMTP client ... 39
5 HTTP(S) ... 39
6 FTP ... 39
7 Web ... 40
Mantis ... 40
BugZilla ... 40
FogBugz ... 40
8 API ... 40
Mantis ... 40
FogBugz ... 41

Part VII All Units 43

1 EAnsiFileNamesFix ... 48
2 EAppCLX ... 49
3 EAppCrossModule ... 49
4 EAppFMX ... 49
5 EAppHTTPApp ... 49
6 EAppType ... 50
Routines ... 50
7 EAppVCL ... 52
8 EAppWebBroker ... 52
9 EBase ... 52
Classes ... 53
Routines ... 53
10 EBaseClasses ... 53

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

11 ECallStack 54
Classes 54
Globals 55
Routines 55
Types 55
12 EClasses 55
Classes 56
13 EClassUtils 56
14 ECompatibility 56
15 EComponent 56
Classes 57
16 EConfig 57
Routines 58
17 EConsts 59
Globals 59
18 ECore 59
19 EDbgHelp 60
20 EDebugDWARF 60
21 EDebugEL 60
22 EDebugEL7 61
23 EDebugExports 61
24 EDebugInfo 62
25 EDebugJCL 62
26 EDebugMap 62
27 EDebugME 63
28 EDebugMS 63
29 EDebugStabs 63
30 EDebugTD32 64
31 EDialog 64
Classes 65
Routines 65
Types 65
32 EDialogCGI 66
33 EDialogConsole 66
34 EDialogHTMLBased 66
35 EDialogISAPI 66
36 EDialogSend 67
37 EDialogSendWinAPI 67
38 EDialogService 67
39 EDialogWeb 67
40 EDialogWER 68
41 EDialogWinAPI 68
42 EDialogWinAPIEurekaLog 69
43 EDialogWinAPIEurekaLogDetailed 69
44 EDialogWinAPIMSClassic 69

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

45 EDialogWinAPIStepsToReproduce ... 70
46 EDisAsm ... 70
Routines ... 70
Types ... 71
47 EDLLs ... 71
48 EDNS ... 71
Classes ... 72
Routines ... 72
49 EEncoding ... 72
50 EEncrypt ... 73
Routines ... 73
Types ... 75
51 EEvents ... 75
Routines ... 77
Types ... 79
52 EException ... 81
Classes ... 82
53 EExceptionHook ... 82
54 EExceptionInfo ... 82
55 EExceptionInfoDelphi2 ... 83
56 EExceptionInfoDelphiANSI ... 83
57 EExceptionInfoDelphiUnicode ... 83
58 EExceptionInfoGeneric ... 83
59 EExceptionInfoMSVC ... 84
60 EExceptionManager ... 84
Classes ... 84
Routines ... 85
Types ... 85
61 EFileMemory ... 85
62 EFixSafeCallException ... 85
63 EFonts ... 85
Routines ... 86
64 EFreeze ... 86
Classes ... 86
Globals ... 87
Routines ... 87
Types ... 87
65 EHash ... 87
Routines ... 88
Types ... 88
66 EHook ... 89
67 EHTMLHelp ... 89
68 EInfoFormat ... 89
69 EInject ... 90
70 EInterfaces ... 90
71 EJSON ... 90
72 EListView ... 90

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

73 ELogBuilder ... 90
Classes ... 91
Globals ... 92
Routines ... 92
Types ... 92
74 ELogging ... 92
75 ELogManager ... 93
76 ELowLevel ... 93
Routines ... 93
77 ELowLevelClasses ... 95
78 EMantisConnectStub ... 95
79 EMAPI ... 95
80 EMapParserBase ... 95
Classes ... 96
81 EMapWin32 ... 96
82 EMemLeaks ... 96
Routines ... 98
Types ... 100
83 EMLang ... 100
84 EModules ... 101
85 EMonitors ... 101
86 ENT ... 101
87 EOSApiList ... 101
88 EPChars ... 102
89 EPEImage ... 102
90 EPNG ... 102
91 EResLeaks ... 102
Routines ... 103
Types ... 103
92 ERunTime ... 103
93 EScreenShot ... 105
94 ESend ... 105
95 ESendAPIBugZilla ... 106
Classes ... 106
96 ESendAPIFogBugz ... 106
Classes ... 107
97 ESendAPIJIRA ... 107
Classes ... 107
98 ESendAPIMantis ... 107
Classes ... 108
99 ESendMail ... 108
100 ESendMailMAPI ... 108
Classes ... 109
101 ESendMailShell ... 109
Classes ... 109
102 ESendMailSMAPI ... 109
Classes ... 110

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

103 ESendMailSMTP ... 110
Classes ... 110
104 ESendWeb ... 111
105 ESendWebFTP ... 111
Classes ... 111
106 ESendWebHTTP ... 111
Classes ... 112
107 ESignature ... 112
Routines ... 112
Types ... 112
108 ESOAP ... 113
109 ESOAPMantis ... 113
110 ESocket ... 113
111 ESockets ... 113
112 ESpecificDelphi ... 114
113 ESSLSocket ... 114
114 EStackTracing ... 114
115 EStrConsts ... 114
116 ESysInfo ... 115
Routines ... 115
117 ETaskDialogs ... 117
118 EThreadsManager ... 117
Classes ... 117
Routines ... 118
Types ... 118
119 ETools ... 118
120 ETypes ... 118
Types ... 119
121 EUnmangling ... 119
122 EWCT ... 119
Routines ... 120
123 EWCTSupport ... 120
124 EWebTools ... 120
125 EWER ... 120
Routines ... 121
126 EWinCrypt ... 121
127 EWinSock ... 121
128 ExceptionLog7 ... 121
Routines ... 122
129 EXML ... 123
130 EXMLBuilder ... 123
131 EXMLRPC ... 123
132 EXMLRPCBugZilla ... 124
133 EZip ... 124
Classes ... 124
Globals ... 124
Routines ... 125

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Types ... 125
134 EZLib ... 125
135 EZLibHeaders ... 125

Part VIII All Classes 128

1 EDNSError ... 130
2 EEurekaLogGeneralError ... 131
3 EZIPException ... 131
4 TBaseDialog ... 131
5 TBaseLogBuilder ... 132
Methods ... 132
Create ... 133
BuildReport ... 133
Refresh ... 134
ToString ... 135
Properties ... 135
FBuilded ... 136
CallStack ... 136
CustomFields ... 137
ExceptionInfo ... 137
Options ... 138
Report ... 138
6 TConsoleWriterDialog ... 139
7 TDummyLogBuilder ... 139
8 TELAbstractMapParser ... 139
Methods ... 139
DecodeIdent ... 140
MapStringToFileName ... 140
MapStringToModuleName ... 141
MapStringToStr ... 142
PrepareStr ... 143
9 TELDebugInfoExports ... 144
10 TELDebugInfoJedi ... 144
11 TELDebugInfoMadExcept ... 144
12 TELDebugInfoMap ... 144
13 TELDebugInfoNative ... 144
14 TELDebugInfoSymbols ... 144
15 TELDebugInfoTD32 ... 144
16 TELFTPSender ... 144
17 TELHTTPSender ... 145
18 TELMailMAPISender ... 145
19 TELMailShellSender ... 145
20 TELMailSimpleMAPISender ... 146
21 TELMailSMTPBaseSender ... 146
22 TELMailSMTPClientSender ... 146
23 TELMailSMTPSender ... 147
24 TELMailSMTPServerSender ... 147
25 TELTrackerBugZillaSender ... 147

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

26 TELTrackerBugZillaSender 147
27 TELTrackerFogBugzSender 148
28 TELTrackerFogBugzSender 148
29 TELTrackerJIRASender 148
30 TELTrackerMantisSender 149
31 TELTrackerMantisSender 149
32 TEurekaBaseStackList 149
Methods 150
GetCurrentInstruction 150
GetCurrentStackFrame 151
GetCurrentStackPointer 152
GetCurrentTopOfStack 152
Create 153
Build 155
33 TEurekaExceptionInfo 157
Methods 157
ToString 157
Properties 158
ActiveControl 160
ActiveControlClass 160
ActiveControlText 161
ActiveForm 161
ActiveFormClass 162
ActiveFormText 162
AdditionalInfo 162
Address 163
BaseException 163
BugID 164
BugIDSource 164
BugIDStr 165
CallStack 165
Context 165
Cursor 166
CursorPos 166
CustomID 167
ExceptionClass 167
ExceptionMessage 167
ExceptionNumber 168
ExceptionObject 168
ExceptionRecord 169
ExpectedContext 169
ExpectedURL 170
Handled 170
Handler 171
Info 171
InitializationException 172
InnerException 172
Module 173
ModuleName 173
ModuleVersion 173
Options 174
Processed 174
Reader 175
ReproduceText 175
SafeCallExpt 176

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

StackDump ... 176
StackPoint ... 177
StackTop ... 177
SynchronizeException ... 178
ThreadException ... 178
ThreadID ... 179
Time ... 179
UnhandledExceptionNumber ... 179
34 TEurekaLogDetailedDialog ... 180
35 TEurekaLogDialog ... 180
36 TEurekaLogEvents ... 180
Events ... 181
OnAttachedFilesRequest ... 182
OnBeginReportGen ... 182
OnCustomBugID ... 183
OnCustomButtonClick ... 183
OnCustomDataRequest ... 183
OnCustomFileName ... 184
OnCustomWebFieldsRequest ... 184
OnEndReportGen ... 184
OnExceptionAction ... 185
OnExceptionError ... 185
OnExceptionNotify ... 186
OnPasswordRequest ... 186
OnRaise ... 186
OnReproduceRequest ... 187
OnZippedFilesRequest ... 187
37 TEurekaModuleOptions ... 188
38 TEurekaStackListV6 ... 188
39 TEurekaStackListV7 ... 189
40 TEurekaThreadData ... 190
Methods ... 190
Priority ... 191
Properties ... 191
CallerAddress ... 192
CallerFunction ... 193
Handle ... 193
ID ... 194
IsMainThread ... 194
IsRTLThread ... 194
IsWindowsThread ... 195
Kind ... 195
Name ... 196
Parameter ... 196
ParentID ... 197
RealParameter ... 197
RealStartingAddress ... 197
RealStartingFunction ... 198
StackTop ... 198
StartingAddress ... 199
StartingFunction ... 199
Thread ... 200
ThreadClassName ... 200
41 TEurekaThreadDataList ... 200
Properties ... 201

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Count ... 201
Items ... 202
ThreadData ... 202
42 TEventLogDialog ... 203
43 TExceptionManager ... 203
Methods ... 203
Instance ... 204
ClearThreadExceptions ... 205
Handle ... 205
Info ... 208
Notify ... 209
Show LastExceptionData ... 211
StandardEurekaError ... 212
StandardEurekaNotify ... 213
Properties ... 214
ExceptionCount ... 215
LastThreadException ... 215
Mode ... 216
UnhandledExceptionCount ... 216
44 TFreezeThread ... 217
Methods ... 217
ChecksPaused ... 218
Execute ... 218
Properties ... 219
TargetThreadHandle ... 219
TargetThreadID ... 219
TargetWindow ... 219
45 TGenericStackList ... 220
Methods ... 221
Create ... 221
Properties ... 222
Tracer ... 222
46 TLogBuilder ... 223
47 TMainThreadFreezeDetectionThread ... 224
48 TMessageBoxDetailedDialog ... 224
49 TMessageBoxDialog ... 224
50 TMSClassicDialog ... 224
51 TNullDialog ... 224
52 TRTLHandlerDialog ... 224
53 TThreadEx ... 225
Methods ... 228
Create ... 228
Execute ... 230
Properties ... 232
AutoHandleException ... 232
54 TWCTFreezeDetectionThread ... 234
55 TWebDialog ... 234
56 TWebDialog ... 234
57 TWebDialog ... 235
58 TWERDialog ... 235
59 TXMLLogBuilder ... 235

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

60 TZipFile ... 235
Methods ... 236
ExtractZipFile ... 236
IsValid ... 237
ZipDirectoryContents ... 237
Create ... 238
Destroy ... 238
Add ... 238
Close ... 240
Extract ... 240
ExtractAll ... 241
IndexOf ... 241
Open ... 242
Read ... 242
Properties ... 243
Comment ... 244
FileComment ... 244
FileCount ... 244
FileInfo ... 245
FileInfos ... 245
FileName ... 245
FileNames ... 245
Mode ... 245

Part IX All Functions 248

1 AllocatePageAlign ... 260
2 AllocMemCountEx ... 261
3 AllocMemCountMax ... 262
4 AllocMemSizeEx ... 262
5 AllocMemSizeMax ... 263
6 AquireThreadData ... 264
7 BeginThreadEx ... 264
8 BuildBugReport ... 267
9 CheckELMagicCode ... 268
10 CheckHeap ... 268
11 ConfigDelete ... 269
12 ConfigFileName ... 270
13 ConfigIsRegistry ... 271
14 ConfigKeyPath ... 272
15 ConfigReadBool ... 273
16 ConfigReadFloat ... 274
17 ConfigReadInteger ... 275
18 ConfigReadString ... 276
19 ConfigWriteBool ... 278
20 ConfigWriteFloat ... 279
21 ConfigWriteInteger ... 280
22 ConfigWriteString ... 281
23 ConvertAddress ... 282
24 CRC16Hash ... 283

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

25 CRC32Hash ... 284
26 CurrentEurekaLogOptions ... 284
27 CurrentInstance ... 286
28 DecodeConfigValue ... 286
29 DecodeInstruction ... 287
30 DisableMemLeaksRegistering ... 288
31 DisasmFunction ... 289
32 DisasmInstruction ... 290
33 DoneCheckFreeze ... 291
34 DumpAllocationsToFile ... 292
35 EnableMemLeaksRegistering ... 294
36 EncodeConfigValue ... 295
37 EncryptBufferFree ... 295
38 EurekaAllocMem ... 297
39 EurekaFreeMem ... 298
40 EurekaGetMem ... 298
41 EurekaReallocMem ... 299
42 EurekaRegisterExpectedMemoryLeak ... 300
43 EurekaRegisterExpectedMemoryLeakDynArray ... 301
44 EurekaRegisterExpectedMemoryLeakStr ... 302
45 EurekaTryGetMem ... 303
46 EurekaUnregisterExpectedMemoryLeak ... 303
47 EurekaUnregisterExpectedMemoryLeakDynArray ... 304
48 EurekaUnregisterExpectedMemoryLeakStr ... 305
49 ExceptionManager ... 305
50 ExpandEnvVars ... 306
51 FindDialogClass ... 307
52 FindLowAddress ... 308
53 GetActiveFormClass ... 308
54 GetActiveFormText ... 309
55 GetApplicationDescription ... 310
56 GetApplicationName ... 310
57 GetApplicationRect ... 311
58 GetApplicationWnd ... 312
59 GetAsmSize ... 312
60 GetCompanyName ... 313
61 GetComputerName ... 314
62 GetControlClass ... 314
63 GetControlText ... 315
64 GetCurrentModuleCompilationDate ... 316
65 GetCurrentUserPrivileges ... 316
66 GetDiskInfo ... 317

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

67 GetDisplayDPI ... 317
68 GetDisplayMode ... 318
69 GetEnvVariable ... 318
70 GetEurekaLogLabel ... 319
71 GetEurekaLogLicense ... 320
72 GetEurekaLogPath ... 321
73 GetEurekaLogStatus ... 322
74 GetEurekaLogVersion ... 322
75 GetExceptionModuleName ... 323
76 GetExceptionModuleVersion ... 324
77 GetFileIcon ... 324
78 GetFileSize ... 325
79 GetFunctionSize ... 326
80 GetMainWnd ... 327
81 GetModuleType ... 327
82 GetMXServerFromEmail ... 328
83 GetReadableSize ... 329
84 GetTaskBarWnd ... 330
85 GetTracer ... 331
86 GetTracerClass ... 332
87 GUIFont ... 333
88 HandleException ... 333
89 HookWin32API ... 334
90 IfElse ... 335
91 IniReadString ... 336
92 InitCheckFreeze ... 337
93 InitSalt ... 338
94 InitWER ... 340
95 IniWriteString ... 340
96 InstructionToString ... 341
97 IntoIDE ... 342
98 IsAppType ... 342
99 IsBPL ... 343
100 IsCGI ... 344
101 IsCGIConsole ... 345
102 IsCGIWin ... 345
103 IsCLX ... 346
104 IsCompiledWithPackages ... 347
105 IsConsole ... 347
106 IsCPL ... 347
107 IsDebugged ... 348
108 IsDLL ... 349

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

109 IsEurekaLogActive ... 349
110 IsEurekaLogActive ... 350
111 IsEurekaLogActiveInThread ... 352
112 IsEurekaLogInstalled ... 353
113 IsEurekaLogInstalled ... 354
114 IsEXE ... 355
115 IsFMX ... 355
116 IsGUI ... 356
117 IsIntraWeb ... 357
118 IsISAPI ... 357
119 IsMemLeaksEnabled ... 358
120 IsMemLeaksInstalled ... 358
121 IsModuleLibrary ... 359
122 IsReadableFile ... 360
123 IsRTLCompiledWithStackFrames ... 360
124 IsService ... 361
125 IsStackFramesProc ... 361
126 IsValidBlockAddr ... 362
127 IsValidClass ... 363
128 IsValidCodeAddr ... 364
129 IsValidHandle ... 365
130 IsValidObject ... 365
131 IsVCL ... 366
132 IsWeb ... 367
133 IsWin10 ... 368
134 IsWin64 ... 368
135 IsWin7 ... 369
136 IsWin8 ... 369
137 IsWin81 ... 370
138 IsWinVista ... 371
139 IsWinXP ... 371
140 IsWorkstation ... 372
141 IsWritableFile ... 373
142 LastBugID ... 373
143 MainThreadTopOfStack ... 374
144 MD5Hash ... 374
145 MemLeaksName ... 375
146 MemLeaksNameDynArray ... 376
147 MemLeaksNameStr ... 376
148 MemLeaksOwn ... 377
149 MemLeaksOwnDynArrayBoth ... 378
150 MemLeaksOwnDynArrayChild ... 378

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

151 MemLeaksOwnDynArrayParent ... 379
152 MemLeaksOwnStr ... 380
153 MemLeaksSetParentBlock ... 380
154 MemLeaksSetParentBlockDynArrayBoth ... 381
155 MemLeaksSetParentBlockDynArrayChild ... 381
156 MemLeaksSetParentBlockDynArrayParent ... 382
157 MemLeaksSetParentBlockStr ... 382
158 MemProtect ... 383
159 MemUnprotect ... 384
160 MonoFont ... 386
161 NameThread ... 386
162 NameThreadForDebugging ... 387
163 PageSize ... 388
164 PauseFreezeCheck ... 388
165 PrimaryInstance ... 389
166 RaiseExpected ... 389
167 RaiseFreezeException ... 390
168 RegisterDialogClass ... 391
169 RegisterDialogClassFirst ... 391
170 RegisterEventAttachedFilesRequest ... 392
171 RegisterEventBeginReportGen ... 393
172 RegisterEventCustomBugID ... 394
173 RegisterEventCustomButtonClick ... 395
174 RegisterEventCustomDataRequest ... 396
175 RegisterEventCustomFileName ... 397
176 RegisterEventCustomWebFieldsRequest ... 398
177 RegisterEventEndReportGen ... 399
178 RegisterEventExceptionAction ... 400
179 RegisterEventExceptionError ... 401
180 RegisterEventExceptionNotify ... 402
181 RegisterEventHelpButtonClick ... 403
182 RegisterEventPasswordRequest ... 404
183 RegisterEventRaise ... 405
184 RegisterEventReproduceRequest ... 406
185 RegisterEventZippedFilesRequest ... 407
186 RegKeyCheck ... 408
187 RegKeyRead ... 408
188 RegKeyWrite ... 409
189 ResumeFreezeCheck ... 410
190 RSADecrypt ... 411
191 RSADestroyKey ... 414
192 RSAEncrypt ... 414

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

193 RSAGenKey ... 417
194 RSALoadPrivateKey ... 418
195 RSALoadPublicKey ... 419
196 RSASavePrivateKey ... 420
197 RSASavePublicKey ... 421
198 RSASign ... 422
199 RSAVerify ... 424
200 SafeExec ... 426
201 SDBMHash ... 427
202 SecureFree ... 428
203 SetAppType ... 429
204 SetEurekaLogState ... 430
205 SetEurekaLogState ... 431
206 SetEurekaLogStateInThread ... 432
207 SetupConfig ... 433
208 SHA1Hash ... 434
209 SHA256Hash ... 434
210 ShowExpected ... 435
211 SignatureGetInfo ... 436
212 TEADecrypt ... 437
213 TEADefaultKey ... 441
214 TEADeriveKey ... 443
215 TEADestroyKey ... 447
216 TEAEncrypt ... 447
217 TEAInitSessionKey ... 451
218 ThreadData ... 452
219 ThreadDataEnum ... 453
220 TwofishDecrypt ... 454
221 TwofishDeriveKey ... 455
222 TwofishDestroyKey ... 457
223 TwofishEncrypt ... 457
224 TwofishInitIV ... 459
225 TwofishInitKey ... 459
226 TZipCompressionToString ... 460
227 UnRegisterEventAttachedFilesRequest ... 461
228 UnRegisterEventBeginReportGen ... 462
229 UnRegisterEventCustomBugID ... 462
230 UnRegisterEventCustomButtonClick ... 463
231 UnRegisterEventCustomDataRequest ... 464
232 UnRegisterEventCustomFileName ... 465
233 UnRegisterEventCustomWebFieldsRequest ... 466
234 UnRegisterEventEndReportGen ... 466

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

235 UnRegisterEventExceptionAction ... 467
236 UnRegisterEventExceptionError ... 468
237 UnRegisterEventExceptionNotify ... 469
238 UnRegisterEventHelpButtonClick ... 470
239 UnRegisterEventPasswordRequest ... 470
240 UnRegisterEventRaise ... 471
241 UnRegisterEventReproduceRequest ... 472
242 UnRegisterEventZippedFilesRequest ... 473
243 WCTDone ... 474
244 WCTInit ... 474

Part X All Types 476

1 PCustomName ... 480
2 PInstruction ... 480
3 PPointerRecord ... 480
4 PSalt ... 481
5 PTwofishInitVector ... 481
6 TAppType ... 482
7 TBaseDialogClass ... 482
8 TBlockType ... 483
btData ... 484
btDynArray ... 484
btObject ... 484
btString ... 484
9 TBugID ... 484
10 TCertificateInfo ... 485
Issuer ... 485
IssuerID ... 486
NotAfter ... 486
NotBefore ... 487
Serial ... 487
Subject ... 488
SubjectID ... 488
Version ... 489
11 TCheckType ... 489
ctEqual ... 490
ctGetLastError ... 491
ctHResult ... 491
ctNone ... 491
ctNotEqual ... 491
ctNTStatus ... 491
12 TCRC16Hash ... 491
13 TCRC32Hash ... 492
14 TCustomName ... 492
15 TDumpCompareFunc ... 493
16 TDumpFilterFunc ... 493
17 TELEvAttachedFilesRequestMeth ... 494
18 TELEvAttachedFilesRequestProc ... 495

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

19 TELEvCustomBugIDMeth ... 496
20 TELEvCustomBugIDProc ... 496
21 TELEvCustomButtonClickMeth ... 497
22 TELEvCustomButtonClickProc ... 498
23 TELEvCustomDataRequestMeth ... 499
24 TELEvCustomDataRequestProc ... 500
25 TELEvCustomFileNameMeth ... 501
26 TELEvCustomFileNameProc ... 501
27 TELEvCustomWebFieldsRequestMeth ... 502
28 TELEvCustomWebFieldsRequestProc ... 503
29 TELEvExceptionActionMeth ... 504
30 TELEvExceptionActionProc ... 505
31 TELEvExceptionErrorMeth ... 507
32 TELEvExceptionErrorProc ... 508
33 TELEvExceptionNotifyMeth ... 510
34 TELEvExceptionNotifyProc ... 511
35 TELEvHelpButtonClickMeth ... 513
36 TELEvHelpButtonClickProc ... 514
37 TELEvPasswordRequestMeth ... 515
38 TELEvPasswordRequestProc ... 516
39 TELEvRaiseMeth ... 517
40 TELEvRaiseProc ... 518
41 TELEvReportGenMeth ... 520
42 TELEvReportGenMeth ... 521
43 TELEvReportGenProc ... 521
44 TELEvReportGenProc ... 522
45 TELEvReproduceRequestMeth ... 523
46 TELEvReproduceRequestProc ... 523
47 TELEvZippedFilesRequestMeth ... 524
48 TELEvZippedFilesRequestProc ... 525
49 TEncryptBuffer ... 525
50 TEurekaActionType ... 526
atCollectedExceptionInfo ... 528
atCollectingExceptionInfo ... 528
atRestarting ... 528
atSavedLogFile ... 529
atSavingLogFile ... 529
atSendCancelled ... 529
atSending ... 529
atSent ... 529
atShowedExceptionInfo ... 530
atShowingExceptionInfo ... 530
atTerminating ... 530
51 TEurekaCallStack ... 530
52 TEurekaCallStackClass ... 531

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

53 TEurekaLogFileType ... 531
ftBugReport ... 532
ftHTMLPage ... 532
ftLog ... 532
ftMiniDump ... 532
ftSavedBugReport ... 533
ftScreenshot ... 533
ftXML ... 533
ftZIP ... 533
ftZIPCopy ... 533
54 TEurekaModuleType ... 533
55 TEurekaStackList ... 534
56 TExceptionNotifyMode ... 534
emD2009 ... 535
emFull ... 536
emHook ... 536
emNull ... 536
emUnknown ... 536
57 TFreezeThreadClass ... 536
58 TInstruction ... 537
Accessible ... 537
Next ... 538
PlatformSpecific ... 538
Start ... 539
Valid ... 539
59 TMD5Hash ... 540
60 TMemoryError ... 540
leHang ... 542
leInvalidBlock ... 542
leMultiFree ... 542
leNone ... 542
leOutOfMemory ... 542
leOverflow ... 543
leUseAfterFree ... 543
61 TMemoryErrors ... 543
62 TModuleType ... 543
mtLibrary ... 544
mtPackage ... 544
mtProgram ... 544
mtUnknown ... 545
63 TPointerRecord ... 545
BlockState ... 546
BlockType ... 546
CallStack ... 546
SingleSize ... 547
64 TResponse ... 547
ErrorCode ... 548
ErrorMessage ... 548
IssueID ... 549
SendResult ... 549
SuccessMessage ... 550
65 TRSAExport ... 551
rsBLOB ... 551

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

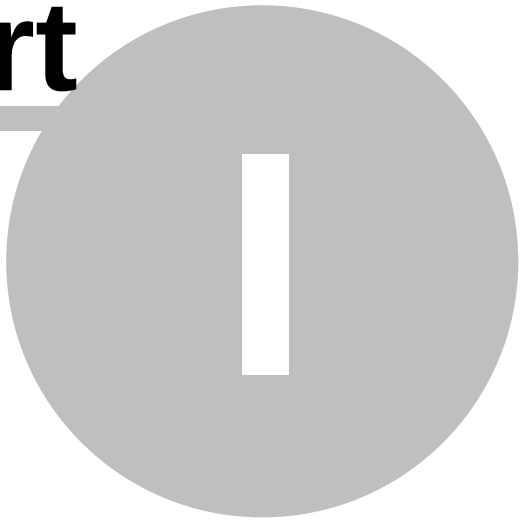
rsDER ... 552
rsPEM ... 552
66 TRSAKey ... 552
PrivateKey ... 553
PublicKey ... 553
67 TRSASignature ... 554
68 TSalt ... 554
69 TSDBMHash ... 555
70 TSendResult ... 555
srBugClosed ... 557
srCancelled ... 557
srConnectionError ... 557
srInvalidInsert ... 558
srInvalidLogin ... 558
srInvalidModify ... 558
srInvalidRead ... 558
srInvalidSearch ... 558
srInvalidSelection ... 559
srNoExceptionInfo ... 559
srObsoleteVersion ... 559
srRestart ... 559
srSent ... 559
srUnknownError ... 560
srUnsupportedSender ... 560
71 TSHA1Hash ... 560
72 TSHA256Hash ... 560
73 TSignatureInfo ... 561
Certificate ... 561
FileName ... 562
MoreInfoLink ... 562
ProgramName ... 563
PublisherLink ... 563
Signed ... 564
Timestamp ... 565
TimestampCertificate ... 565
Timestamped ... 566
74 TStreamConstructor ... 567
75 TTEAKey ... 567
76 TThreadKind ... 568
tkBeginThread ... 568
tkGeneric ... 568
tkTThread ... 569
77 TTwofishInitVector ... 569
78 TTwofishKey ... 569
79 TTwofishRAWKey ... 570
80 TZipCompression ... 570
zcBZIP2 ... 572
zcDeflate ... 572
zcDeflate64 ... 572
zcImplode ... 573
zcLZ77 ... 573
zcLZMA ... 573
zcPKImplode ... 573

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

zcPPMdI1 ... 573
zcReduce1 ... 574
zcReduce2 ... 574
zcReduce3 ... 574
zcReduce4 ... 574
zcReserved11 ... 574
zcReserved13 ... 575
zcReserved15 ... 575
zcReserved16 ... 575
zcReserved17 ... 575
zcShrunk ... 575
zcStored ... 576
zcTERSE ... 576
zcTokenize ... 576
zcWavePack ... 576
81 TZipEndOfCentralHeader ... 576
82 TZipHeader ... 577
HostOS ... 578
MadeByVersion ... 578
RequiredVersion ... 579
Part XI All Globals 582
1 DefaultTracerMask ... 582
2 EEurekaLogCurrentVersion ... 583
3 EEurekaLogLicense ... 583
4 EEurekaLogVersion ... 584
5 EurekaCallStackClass ... 584
6 FreezeThreadClass ... 585
7 LogBuilderClass ... 586
8 CENTRALHEADER ... 586
9 DATADESCRIPTOR ... 587
10 LOCALHEADER ... 587
11 ZIPENDOFHEADER ... 588
Index 589

© 2001-2015, 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 305
- Exception information 157
- Call stack functions 54
- Location routines 62
- Informational functions 115
- Formatting functions 89
- Module functions 101
- Thread functions 117
- Screenshot functions 105
- More functions 248

1.1 IsEurekaLogInstalled

Indicates if EurekaLog is installed in this application.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
function IsEurekaLogInstalled: Boolean;
```

Return value

True if EurekaLog is installed, False - otherwise

© 2001-2015, 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 121);
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 354 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 121

Syntax

Code (Delphi)

```
function IsEurekaLogActive: Boolean;
```

© 2001-2015, 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 354 function.

Note

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

Note

This is a wrapper for IsEurekaLogActive 349 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

1.3 IsEurekaLogActiveInThread

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

Unit

ExceptionLog7 121

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- SetEurekaLogStateInThread 7
- _InstallEurekaLog
- _IsEurekaLogActive

1.4 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
procedure SetEurekaLogState(
   const AActivate: 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 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 431 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- SetEurekaLogStateInThread 7

1.5 SetEurekaLogStateInThread

Enables or disables EurekaLog in the specified thread.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
procedure SetEurekaLogStateInThread(
   const AThreadID: Cardinal;
   const AActivate: 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.

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)

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

See also

- IsEurekaLogActive 3

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- SetEurekaLogState 6

1.6 CurrentEurekaLogOptions

Returns current EurekaLog's options.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
function CurrentEurekaLogOptions: TEurekaModuleOptions 188;
```

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 188 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 121

Syntax

Code (Delphi)

```
function LastBugID: TBugID 484;
```

Return value

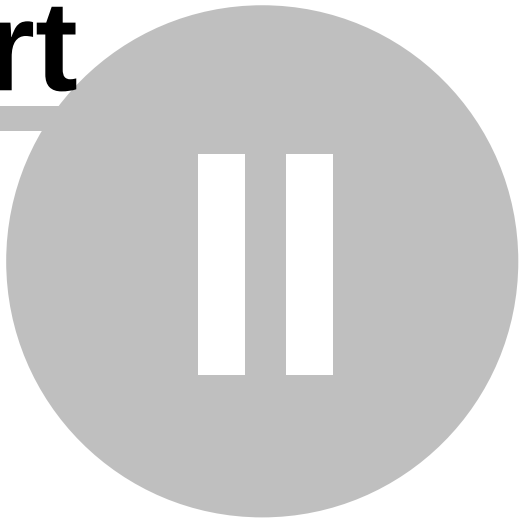
BugID of last thread exception or 0 if there was no exception raised.

See also

- BugID

© 2001-2015, 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 15	Fires for exception processing stages; allows you to fine-control exception handling
OnExceptionError 17	Fires on any error during exception processing
OnPasswordRequest 18	Use this event to supply a password to *decrypt* call stacks
OnAttachedFilesRequest 19	Include additional files into bug report (separate files)
OnZippedFilesRequest 20	Include additional files into zipped bug report
OnReproduceRequest 21	Supply a custom steps-to-reproduce data
OnCustomFileName 21	Allows you to alter any file name
OnCustomBugID 22	Customize BugID by supplying sub-ID or replace it completely with your own values
OnCustomDataRequest 23	Include additional data into bug report (text)
OnCustomWebFieldRequest 23	Customize request to web server or bug tracker with additional custom data
OnCustomButtonClick 24	Fires when "Custom" button is clicked in error dialog
OnHelpButtonClick 25	Fires when "Help" button is clicked in error dialog
OnBeginReportGen/OnEndReportGen 26	Fires before and after bug report generating

A typical order of events may go like this:

- Raise stage:
  - OnExceptionAction atCollectingExceptionInfo
  - OnRaise
  - OnExceptionAction atCollectedExceptionInfo
  - **Debugger notification about exception**

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- 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 29
- Functions 2

2.1 OnRaise

Method event handler for OnRaise event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvRaiseMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

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 75 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 405
- Procedural OnRaise event handler 518
- EurekaLog component 29
- General events concept 75
- Other events 11

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

2.2 OnExceptionNotify

Method event handler for OnExceptionNotify event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionNotifyMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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

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).

Tip

You can also use exception filters instead of this event.

Use OnExceptionAction 15 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.

© 2001-2015, 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 handled b
```

See this article for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure MyObject.OnExceptionNotify(AExceptionInfo: TEurekaExceptionInf
begin
    // Disable exception processing by EurekaLog
    Handle := False;
    ACallNextHandler := False;
end;
```

See also

- Event handler registration 402
- Procedural OnExceptionNotify event handler 511
- EurekaLog component 29
- General events concept 75
- Other events 11

2.3 OnExceptionAction

Method event handler for OnExceptionAction event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionActionMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AAdditionalInfo: String;
   var AExecute: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AEurekaAction [in]
Action to perform.

AAdditionalInfo [in]
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 successfull, `False` - action was unsuccessfull;

> **Note**
>
> This argument is ignored for atSendCancelled 529.

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

> **Tip**
>
> Set to `False`, if you set 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 529 and atSent 529 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 37.

> **Note**
>
> 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" ...);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
          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)

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

See also

- Event handler registration 400
- Procedural OnExceptionAction event handler 505
- EurekaLog component 29
- General events concept 75
- Other events 11

2.4 OnExceptionError

Method event handler for OnExceptionError event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionErrorMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
   const AAdditionalInfo: String;
   var ARetry: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Information about exception.

AEurekaAction [in]
Failed action.

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

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.

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.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 529 and atSent 529 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 37.

See OnExceptionAction 15 for more information.

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.

See also

- Event handler registration 401
- Procedural OnExceptionError event handler 508
- EurekaLog component 29
- General events concept 75
- Other events 11

2.5 OnPasswordRequest

Method event handler for OnPasswordRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AModule [in]
Module handle which password is retrieved.

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 values. 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.

See also

- Event handler registration 404
- Method OnPasswordRequest event handler 516
- EurekaLog component 29
- General events concept 75
- Other events 11

2.6 OnAttachedFilesRequest

Method event handler for OnAttachedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

ATempFolder [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Temp folder name with trailing path delimiter. You can use it to store temp any 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 392
- Procedural OnAttachedFilesRequest event handler 495
- OnZippedFilesRequest event handler 20
- EurekaLog component 29
- General events concept 75
- Other events 11

2.7 OnZippedFilesRequest

Method event handler for OnZippedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 407
- Procedural OnZippedFilesRequest event handler 525
- OnAttachedFilesRequest event handler 19
- EurekaLog component 29
- General events concept 75
- Other events 11

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

2.8 OnReproduceRequest

Method event handler for OnReproduceRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvReproduceRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   var AReproduceText: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 406
- Procedural OnReproduceRequest event handler 523
- EurekaLog component 29
- General events concept 75
- Other events 11

2.9 OnCustomFileName

Method event handler for OnCustomFileName event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomFileNameMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AFileType: TEurekaLogFileType 531;
   var AFileName: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AFileType [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 397
- Procedural OnCustomFileName event handler 501
- EurekaLog component 29
- General events concept 75
- Other events 11

2.10 OnCustomBugID

Method event handler for OnCustomBugID event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomBugIDMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

generated by EurekaLog.

See also

- Event handler registration 394
- Procedural OnCustomBugID event handler 496
- EurekaLog component 29
- General events concept 75
- Other events 11

2.11 OnCustomDataRequest

Method event handler for OnCustomDataRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomDataRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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

See also

- Event handler registration 396
- Procedural OnCustomDataRequest event handler 500
- EurekaLog component 29
- General events concept 75
- Other events 11

2.12 OnCustomWebFieldsRequest

Method event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 75

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

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

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 398
- Procedural OnCustomWebFieldsRequest event handler 503
- EurekaLog component 29
- General events concept 75
- Other events 11

2.13 OnCustomButtonClick

Method event handler for OnCustomButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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

© 2001-2015, 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 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 25.
- 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 395
- Procedural OnCustomButtonClick event handler 498
- OnHelpButtonClick event handler 25
- EurekaLog component 29
- General events concept 75
- Other events 11

2.14 OnHelpButtonClick

Method event handler for OnHelpButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvHelpButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

Remarks

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 24.

See also

- Event handler registration 403
- Procedural OnHelpButtonClick event handler 514
- OnCustomButtonClick event handler 24
- EurekaLog component 29
- General events concept 75
- Other events 11
- RaiseExpected routine 389

2.15 OnBeginReportGen/OnEndReportGen

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

EEvents 75

Syntax

Code (Delphi)

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

Parameters

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 132 instance. You need to use type cast.

Note
A more convenient case of this type 521 is declared in ELogBuilder 90 unit. Use it instead of this type whenever possible. Type from ELogBuilder doesn't require you to use typecast.

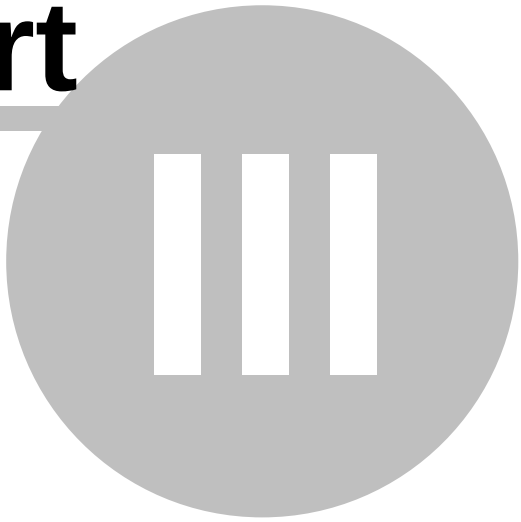
© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration
- Procedural OnReportBeginGen/OnReportEndGen event handler 17
- EurekaLog component 29
- General events concept 75
- Other events 11

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



3 Component

EurekaLog 7 component.

Unit

EComponent 56

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 75 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 75 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 30	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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

3.1 Events

OnAttachedFilesRequest 182	Wrapper for OnAttachedFilesRequest 19 event.
OnBeginReportGen 182	Wrapper for OnBeginReportGen 26 event.
OnCustomBugID 183	Wrapper for OnCustomBugID 22 event.
OnCustomButtonClick 183	Wrapper for OnCustomButtonClick 24 event.
OnCustomDataRequest 183	Wrapper for OnCustomDataRequest 23 event.
OnCustomFileName 184	Wrapper for OnCustomFileName 21 event.
OnCustomWebFieldsRequest 184	Wrapper for OnCustomWebFieldsRequest 23 event.
OnEndReportGen 184	Wrapper for OnEndReportGen 26 event.
OnExceptionAction 185	Wrapper for OnExceptionAction 15 event.
OnExceptionError 185	Wrapper for OnExceptionError 17 event.
OnExceptionNotify 186	Wrapper for OnExceptionNotify 14 event.
OnPasswordRequest 186	Wrapper for OnPasswordRequest 18 event.
OnRaise 186	Wrapper for OnRaise 12 event.
OnReproduceRequest 187	Wrapper for OnReproduceRequest 21 event.
OnZippedFilesRequest 187	Wrapper for OnZippedFilesRequest 20 event.

See also

- TEurekaLogEvents 29
- EComponent 56

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



4 Dialogs

Base class 32	Base class for all dialogs
Null dialog 33	Dialog class which does nothing
RTL dialog 33	Dialog class which executes standard exception processing
Console 33	Dialog class for console applications
MessageBox 33	Simple dialog class for GUI applications
MS Classic 35	Main dialog class for GUI applications
EurekaLog 33	Debug dialog class for GUI applications
CGI 33	Dialog class for CGI applications
ISAPI 33	Dialog class for ISAPI applications
IntraWeb 33	Dialog class for IntraWeb applications
WER 33	Dialog class which invokes OS exception processing
System Log 33	Dialog class for NT services

See also

- Dialogs description

4.1 Base class

Base class for all dialogs.

Unit

EDialog 64

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 391 or RegisterDialogClassFirst 391 functions.

© 2001-2015, 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part V

5 Debug information providers

EurekaLog 35	Build-in class for EurekaLog format
MAP 35	Provider class for .map files
TD32 35	Provider class for .tds files (Turbo Debugger)
JEDI 35	Provider class for JclDebug format
madExcept 35	Provider class for madExcept format
Microsoft 35	Provider class for Microsoft formats
DLL export table 35	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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part VI

6 Send engines

mailto 37	Send class for "mailto" protocol (send via Shell)
Simple MAPI 37	Send class for Simple MAPI protocol
MAPI 38	Send class for MAPI protocol
SMTP 38	Send class for SMTP protocol
HTTP(S) 39	Send class for HTTP/HTTPS protocols
FTP 39	Send class for FTP protocol
Mantis 40	Send class for Mantis bug tracker
FogBugz 41	Send class for FogBugz bug tracker
BugZilla 147	Send class for BugZilla bug tracker
JIRA 148	Send class for JIRA bug tracker

See also

- Send engines description

6.1 mailto

Implements sending via mailto: protocol

Unit

ESendMailShell 109

Syntax

Code (Delphi)

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

Remarks

Works via ShellExecute.

6.2 Simple MAPI

Implements sending via Simple MAPI

Unit

ESendMailSMAPI 109

© 2001-2015, 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 108

Syntax

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

Remarks

Implemented by Outlook.

6.4 SMTP

Unit

ESendMailSMTP 110

Syntax

```
Code (Delphi)
TELMailSMTPSender = class(TELMailSMTPBaseSender 146);
```

Remarks

6.4.1 SMTP server

Implements sending via SMTP server protocol.

Unit

ESendMailSMTP 110

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TELMailSMTPServerSender = class(TELMailSMTPBaseSender 146);
```

Remarks

Works by creating a fake e-mail server.

6.4.2 SMTP client

Implements sending via SMTP client protocol.

Unit

ESendMailSMTP 110

Syntax

```
Code (Delphi)
TELMailSMTPClientSender = class(TELMailSMTPBaseSender 146);
```

Remarks

Same as normal e-mail client application.

6.5 HTTP(S)

Implements sending via HTTP posting.

Unit

ESendWebHTTP 111

Syntax

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

Remarks

Requires a custom receiver script or form.

6.6 FTP

Implements sending via FTP uploading.

Unit

ESendWebFTP 111

© 2001-2015, 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 40	Send class for Mantis bug tracker (web-based)
FogBugz 40	Send class for FogBugz bug tracker (web-based)
BugZilla 40	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 40	Send class for Mantis bug tracker (API-access)
FogBugz 41	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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

ESendAPIMantis 107

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 106

Syntax

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

Remarks

Fogbugz uses custom XML API.

Requirements

Bug tracker	Fogbugz 6 or above

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



7 All Units

EAnsiFileNamesFix 48	This is unit EAnsiFileNamesFix
EAppCLX 49	This is unit EAppCLX
EAppCrossModule 49	This is unit EAppCrossModule
EAppFMX 49	This is unit EAppFMX
EAppHTTPApp 49	This is unit EAppHTTPApp
EAppType 50	General support unit for application type concept (low-level).
EAppVCL 52	This is unit EAppVCL
EAppWebBroker 52	This is unit EAppWebBroker
EBase 52	This is a basic unit for EurekaLog.
EBaseClasses 53	This is unit EBaseClasses
ECallStack 54	Unit for call stack classes and tracing code.
EClasses 55	This is unit EClasses
EClassUtils 56	This is unit EClassUtils
ECompatibility 56	Language forward compatibility support unit.
EComponent 56	This unit contains EurekaLog 7 component.
EConfig 57	Configuration management unit.
EConsts 59	Global constants.
ECore 59	This is unit ECore
EDbgHelp 60	This is unit EDbgHelp
EDebugDWARF 60	This is unit EDebugDWARF
EDebugEL 60	This is unit EDebugEL
EDebugEL7 61	This is unit EDebugEL7

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EDebugExports 61	This is unit `EDebugExports`
EDebugInfo 62	This is unit `EDebugInfo`
EDebugJCL 62	This is unit `EDebugJCL`
EDebugMap 62	This is unit `EDebugMap`
EDebugME 63	This is unit `EDebugME`
EDebugMS 63	This is unit `EDebugMS`
EDebugStabs 63	This is unit `EDebugStabs`
EDebugTD32 64	This is unit `EDebugTD32`
EDialog 64	Base unit for dialogs.
EDialogCGI 66	This is unit `EDialogCGI`
EDialogConsole 66	This is unit `EDialogConsole`
EDialogHTMLBased 66	This is unit `EDialogHTMLBased`
EDialogISAPI 66	This is unit `EDialogISAPI`
EDialogSend 67	This is unit `EDialogSend`
EDialogSendWinAPI 67	This is unit `EDialogSendWinAPI`
EDialogService 67	This is unit `EDialogService`
EDialogWeb 67	This is unit `EDialogWeb`
EDialogWER 68	This is unit `EDialogWER`
EDialogWinAPI 68	This is unit `EDialogWinAPI`
EDialogWinAPIEurekaLog 69	This is unit `EDialogWinAPIEurekaLog`
EDialogWinAPIEurekaLogDetailed 69	This is unit `EDialogWinAPIEurekaLogDetailed`
EDialogWinAPIMSClassic 69	This is unit `EDialogWinAPIMSClassic`
EDialogWinAPIStepsToReproduce 70	This is unit

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	EDialogWinAPIStepsToReproduce
EDisAsm 70	Disassembler unit.
EDLLs 71	Support unit for dynamic DLL loading.
EDNS 71	DNS support unit.
EEncoding 72	Encoding support unit.
EEncrypt 73	Encryption support unit.
EEvents 75	EurekaLog events.
EException 81	Unit with exception information class.
EExceptionHook 82	This is unit EExceptionHook
EExceptionInfo 82	This is unit EExceptionInfo
EExceptionInfoDelphi2 83	This is unit EExceptionInfoDelphi2
EExceptionInfoDelphiANSI 83	This is unit EExceptionInfoDelphiANSI
EExceptionInfoDelphiUnicode 83	This is unit EExceptionInfoDelphiUnicode
EExceptionInfoGeneric 83	This is unit EExceptionInfoGeneric
EExceptionInfoMSVC 84	This is unit EExceptionInfoMSVC
EExceptionManager 84	No description available
EFileMemory 85	This is unit EFileMemory
EFixSafeCallException 85	This is unit EFixSafeCallException
EFonts 85	Adaptive fonts support unit.
EFreeze 86	Freeze detection unit.
EHash 87	Hashing support unit.
EHook 89	This is unit EHook

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EHTMLHelp 89	This is unit EHTMLHelp
EInfoFormat 89	This is unit EInfoFormat
EInject 90	This is unit EInject
EInterfaces 90	This is unit EInterfaces
EJSON 90	This is unit EJSON
EListView 90	This is unit EListView
ELogBuilder 90	Bug report builder unit.
ELogging 92	This is unit ELogging
ELogManager 93	This is unit ELogManager
ELowLevel 93	Low level routines.
ELowLevelClasses 95	Low-level classes support unit.
EMantisConnectStub 95	This is unit EMantisConnectStub
EMAPI 95	This is unit EMAPI
EMapParserBase 95	This is unit EMapParserBase
EMapWin32 96	This is unit EMapWin32
EMemLeaks 96	Unit for diagnosing memory problems (leaks, overflows, etc.).
EMLang 100	MLang headers.
EModules 101	This is unit EModules
EMonitors 101	This is unit EMonitors
ENT 101	This is unit ENT
EOSApiList 101	This is unit EOSApiList
EPChars 102	This is unit EPChars
EPEImage 102	PE(64) formats support unit.
EPNG 102	This is unit EPNG

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EResLeaks 102	This is unit EResLeaks
ERunTime 103	This is unit ERunTime
EScreenShot 105	This is unit EScreenShot
ESend 105	This is unit ESend
ESendAPIBugZilla 106	This is unit ESendAPIBugZilla
ESendAPIFogBugz 106	This is unit ESendAPIFogBugz
ESendAPIJIRA 107	This is unit ESendAPIJIRA
ESendAPIMantis 107	This is unit ESendAPIMantis
ESendMail 108	This is unit ESendMail
ESendMailMAPI 108	This is unit ESendMailMAPI
ESendMailShell 109	This is unit ESendMailShell
ESendMailSMAPI 109	This is unit ESendMailSMAPI
ESendMailSMTP 110	This is unit ESendMailSMTP
ESendWeb 111	This is unit ESendWeb
ESendWebFTP 111	This is unit ESendWebFTP
ESendWebHTTP 111	This is unit ESendWebHTTP
ESignature 112	Unit for working with digital signatures of executable files.
ESOAP 113	This is unit ESOAP
ESOAPMantis 113	This is unit ESOAPMantis
ESocket 113	This is unit ESocket
ESockets 113	This is unit ESockets
ESpecificDelphi 114	This is unit ESpecificDelphi
ESSLSocket 114	This is unit ESSLSocket
EStackTracing 114	Low-level stack tracing code.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EStrConsts 114	String constants.
ESysInfo 115	System information functions.
ETaskDialogs 117	This is unit `ETaskDialogs`
EThreadsManager 117	EurekaLog thread manager unit.
ETools 118	This is unit `ETools`
ETypes 118	Basic EurekaLog types.
EUnmangling 119	This is unit `EUnmangling`
EWCT 119	WCT headers.
EWCTSupport 120	This is unit `EWCTSupport`
EWebTools 120	This is unit `EWebTools`
EWER 120	WER headers.
EWinCrypt 121	Windows cryptography headers.
EWinSock 121	WinSock headers.
ExceptionLog7 121	Main EurekaLog 7 unit.
EXML 123	This is unit `EXML`
EXMLBuilder 123	This is unit `EXMLBuilder`
EXMLRPC 123	This is unit `EXMLRPC`
EXMLRPCBugZilla 124	This is unit `EXMLRPCBugZilla`
EZip 124	Compression support unit.
EZLib 125	Compression support unit.
EZLibHeaders 125	This is unit `EZLibHeaders`

7.1 EAnsiFileNamesFix

This is unit `EAnsiFileNamesFix`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EHook Unit 89
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETools Unit 118
- ETypes Unit 118

7.2 EAppCLX

This is unit EAppCLX

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.3 EAppCrossModule

This is unit EAppCrossModule

See also

- EInterfaces Unit 90
- ETypes Unit 118

7.4 EAppFMX

This is unit EAppFMX

See also

- EAppType Unit 50
- EBase Unit 52
- EConsts Unit 59
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.5 EAppHTTPApp

This is unit EAppHTTPApp

See also

- EAppType Unit 50
- EBase Unit 52
- EConsts Unit 59
- EException Unit 81
- EExceptionHook Unit 82
- EHook Unit 89

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.6 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 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 50
- EConsts Unit 59
- EDLLs Unit 71
- ELogging Unit 92
- ELowLevel Unit 93
- ETaskDialogs Unit 117
- ETypes Unit 118

7.6.1 Routines

GetApplicationWnd 312	Returns application window (if applicable).
GetMainWnd 327	Returns main window of application (if applicable).
GetModuleType 327	Returns type of loaded executable module, based on analyzing its file header.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

GetTaskBarWnd 330	Returns window of application, which creates a taskbar button (if applicable).
IsAppType 342	Checks if application is of specified type.
IsBPL 343	Checks if specified module is BPL file.
IsCGI 344	Checks if application is CGI application.
IsCGIConsole 345	Checks if application is CGI console application.
IsCGIWin 345	Checks if application is CGI Windows application.
IsCLX 346	Checks if application is CLX application.
IsConsole 347	Checks if application is console application.
IsCPL 347	Checks if application is control panel application.
IsDLL 349	Checks if specified module is DLL file.
IsEXE 355	Checks if specified module is EXE file.
IsFMX 355	Checks if application is FMX application.
IsGUI 356	Checks if application is GUI application.
IsIntraWeb 357	Checks if application is IntraWeb application.
IsISAPI 357	Checks if application is ISAPI application.
IsModuleLibrary 359	Checks if specified module is DLL or BPL file.
IsService 361	Checks if application is a win32 service application.
IsVCL 366	Checks if application is VCL application.
IsWeb 367	Checks if application is web application.
SetAppType 429	Set a type of this application.

See also

- EAppType Unit 50

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.7 EAppVCL

This is unit EAppVCL

See also

- EAppType Unit 50
- EBase Unit 52
- EConsts Unit 59
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- ETypes Unit 118

7.8 EAppWebBroker

This is unit EAppWebBroker

See also

- EAppHTTPApp Unit 49
- EAppType Unit 50
- EBase Unit 52
- EConsts Unit 59
- EException Unit 81
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.9 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)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

See also

- ExceptionLog7 121
- ExceptionLog
- EBase Classes 53
- EBase Routines 53
- EAppType Unit 50
- EConsts Unit 59
- ELogging Unit 92
- ELowLevel Unit 93
- ETypes Unit 118

7.9.1 Classes

EEurekaLogGeneralError 131	No description available
TThreadEx 225	Abstract class to create separate threads of execution.

See also

- EBase Unit 52

7.9.2 Routines

BeginThreadEx 264	Spawns a separate thread of execution.
HandleException 333	Handles exception.
IsEurekaLogActive 349	Indicates if EurekaLog is active or disabled in this application.
IsEurekaLogInstalled 354	Indicates if EurekaLog is installed in this application.
RaiseExpected 389	Raises expected exception.
SetEurekaLogState 431	Enables or disables EurekaLog in this application.
ShowExpected 435	Shows expected exception.

See also

- EBase Unit 52

7.10 EBaseClasses

This is unit `EBaseClasses`

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EAppType Unit 50
- EBase Unit 52
- ECompatibility Unit 56
- EDLLs Unit 71
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETools Unit 118
- ETypes Unit 118

7.11 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 114
- EDebugInfo unit 62
- ECallStack Classes 54
- ECallStack Types 55
- ECallStack Routines 55
- ECallStack Globals 55
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDisAsm Unit 70
- EDlls Unit 71
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- ENT Unit 101
- EResLeaks Unit 102
- EStrConsts Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETypes Unit 118
- EWCT Unit 119
- EWCTSupport Unit 120

7.11.1 Classes

TEurekaBaseStackList 149	Base call stack class.
TEurekaStackListV6 188	Stack tracing class for EurekaLog 6 method.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TEurekaStackListV7 189	Stack tracing class for EurekaLog 7 RAW method.
TGenericStackList 220	Common stack tracing class.

See also
- ECallStack Unit 54

7.11.2 Globals

DefaultTracerMask 582	Mask for default call stack class.
EurekaCallStackClass 584	Default call stack class.

See also
- ECallStack Unit 54

7.11.3 Routines

GetTracer 331	Get stack tracing class by number.
GetTracerClass 332	Get stack tracing class by number.

See also
- ECallStack Unit 54

7.11.4 Types

TEurekaCallStack 530	Alias for TEurekaStackListV7.
TEurekaCallStackClass 531	Class of TEurekaBaseStackList.
TEurekaStackList 534	Alias for TEurekaStackListV7.

See also
- ECallStack Unit 54

7.12 EClasses

This is unit `EClasses`

See also
- EClasses Classes 56
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- EExceptionInfo Unit 82
- EHash Unit 87
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- EStackTracing Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118

7.12.1 Classes

TEurekaModuleOptions 188	EurekaLog options.

See also

- EClasses Unit 55

7.13 EClassUtils

This is unit `EClassUtils`

See also

- ELowLevel Unit 93
- ETypes Unit 118

7.14 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 53
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.15 EComponent

This unit contains EurekaLog 7 component.

Remarks

EurekaLog component is used as simple way to react to events. This is analog of TApplicationEvents

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

component. It is optional, it is not requited. See description of the component for more info.

See also

- TEurekaLogEvents 29
- Events 11
- EComponent Classes 57
- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- EDialog Unit 64
- EEvents Unit 75
- EException Unit 81
- ELogBuilder Unit 90
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ESend Unit 105
- ESendWeb Unit 111
- ETypes Unit 118

7.15.1 Classes

TEurekaLogEvents 29	EurekaLog 7 component.

See also

- EComponent Unit 56

7.16 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 276 function. Similary, all ConfigWriteXXX procedures are just wrappers around ConfigWriteString 281 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 433 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 336/IniWriteString 340, backward-compatibility routines ReadRawStrings/WriteRawStrings and also a helper functions like EncodeConfigValue 295/DecodeConfigValue 286 and informational routines ConfigIsRegistry 271, ConfigFileName 270, ConfigKeyPath 272.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EConfig Routines 58
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- EEncrypt Unit 73
- EHash Unit 87
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESysInfo Unit 115
- ETypes Unit 118

7.16.1 Routines

ConfigDelete 269	Deletes a key from the specified section.
ConfigFileName 270	Returns .ini-file name to store EurekaLog configuration.
ConfigIsRegistry 271	Indicates if EurekaLog configuration is stored in the registry or in .ini-file.
ConfigKeyPath 272	Returns registry key name to store EurekaLog configuration.
ConfigReadBool 273	Reads boolean value from specified key and section.
ConfigReadFloat 274	Reads float value from specified key and section.
ConfigReadInteger 275	Reads integer value from specified key and section.
ConfigReadString 276	Reads string value from specified key and section.
ConfigWriteBool 278	Writes boolean value to specified key and section.
ConfigWriteFloat 279	Writes float value to specified key and section.
ConfigWriteInteger 280	Writes integer value to specified key and section.
ConfigWriteString 281	Writes string value to specified key and section.
DecodeConfigValue 286	Low-level routine to decode data after reading from configuration storage.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EncodeConfigValue 295	Low-level routine to encode data before saving to configuration storage.
IniReadString 336	Reads string from .ini file.
IniWriteString 340	Writes string to .ini file.
RegKeyCheck 408	Checks registry key for existing.
RegKeyRead 408	Reads string from registry key.
RegKeyWrite 409	Writes string to registry key.
SetupConfig 433	Changes EurekaLog configuration storage.

See also

- EConfig Unit 57

7.17 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 114
- EConsts Globals 59

7.17.1 Globals

EEurekaLogCurrentVersion 583	Current EurekaLog version (numeric).
EEurekaLogLicense 583	No description available
EEurekaLogVersion 584	Current EurekaLog version (textual).

See also

- EConsts Unit 59

7.18 ECore

This is unit `ECore`

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EInfoFormat Unit 89
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- EStrConsts Unit 114
- ETypes Unit 118

7.19 EDbgHelp

This is unit EDbgHelp

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71
- ELowLevel Unit 93
- EPEImage Unit 102
- ETypes Unit 118

7.20 EDebugDWARF

This is unit EDebugDWARF

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EEncoding Unit 72
- ELowLevel Unit 93
- EModules Unit 101
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EZLib Unit 125

7.21 EDebugEL

This is unit EDebugEL

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugEL7 Unit 61

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EDebugInfo Unit 62
- EEncoding Unit 72
- EEncrypt Unit 73
- EEvents Unit 75
- EFileMemory Unit 85
- EHash Unit 87
- ELogging Unit 92
- ELowLevel Unit 93
- EModules Unit 101
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118
- EZLib Unit 125

7.22 EDebugEL7

This is unit `EDebugEL7`

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EEncoding Unit 72
- EEncrypt Unit 73
- EEvents Unit 75
- EFileMemory Unit 85
- ELogging Unit 92
- ELowLevel Unit 93
- EModules Unit 101
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EZLib Unit 125

7.23 EDebugExports

This is unit `EDebugExports`

See also

- EBaseClasses Unit 53
- ECallStack Unit 54
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EDisAsm Unit 70
- EFileMemory Unit 85
- ELogging Unit 92
- ELowLevel Unit 93
- EPEImage Unit 102
- EStrConsts Unit 114

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ETools Unit 118
- ETypes Unit 118
- EUnmangling Unit 119

7.24 EDebugInfo

This is unit `EDebugInfo`

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- EThreadsManager Unit 117
- ETypes Unit 118

7.25 EDebugJCL

This is unit `EDebugJCL`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- ELogging Unit 92
- ELowLevel Unit 93
- EPEImage Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118

7.26 EDebugMap

This is unit `EDebugMap`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- ELogging Unit 92
- ELowLevel Unit 93
- EMapParserBase Unit 95
- EPEImage Unit 102
- EStrConsts Unit 114
- ETools Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 118

7.27 EDebugME

This is unit EDebugME

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EEncoding Unit 72
- ELowLevel Unit 93
- EModules Unit 101
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EZLib Unit 125

7.28 EDebugMS

This is unit EDebugMS

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDbgHelp Unit 60
- EDebugEL Unit 60
- EDebugInfo Unit 62
- EDLLs Unit 71
- ELogging Unit 92
- ELowLevel Unit 93
- EModules Unit 101
- EPEImage Unit 102
- EStrConsts Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETools Unit 118
- ETypes Unit 118

7.29 EDebugStabs

This is unit EDebugStabs

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EEncoding Unit 72
- ELowLevel Unit 93
- EModules Unit 101

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EPEImage Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EZLib Unit 125

7.30 EDebugTD32

This is unit EDebugTD32

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- ELogging Unit 92
- ELowLevel Unit 93
- EPEImage Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EUnmangling Unit 119

7.31 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 68
- EDialogSendWinAPI 67
- EDialogHTMLBased 66
- EDialogCGI 66
- EDialogConsole 66
- EDialogISAPI 66
- EDialogService 67
- EDialogWeb 67
- EDialogWER 68
- EDialogSend 67
- EDialog Classes 65
- EDialog Types 65
- EDialog Routines 65
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EDLLs Unit 71
- EEncoding Unit 72
- EEvents Unit 75
- EException Unit 81
- EHash Unit 87
- EInfoFormat Unit 89
- ELogBuilder Unit 90
- ELogging Unit 92
- ELogManager Unit 93
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- EScreenshot Unit 105
- ESend Unit 105
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETaskDialogs Unit 117
- EThreadsManager Unit 117
- ETools Unit 118
- ETypes Unit 118
- EZip Unit 124

7.31.1 Classes

TBaseDialog 32	Base class for all dialogs.

See also
- EDialog Unit 64

7.31.2 Routines

FindDialogClass 307	Searches dialog class by name.
RegisterDialogClass 391	Registers dialog class.
RegisterDialogClassFirst 391	Registers dialog class.

See also
- EDialog Unit 64

7.31.3 Types

TBaseDialogClass 482	Pointer to dialog class.

See also
- EDialog Unit 64

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.32 EDialogCGI

This is unit `EDialogCGI`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDialog Unit 64
- EDialogWeb Unit 67
- EException Unit 81
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.33 EDialogConsole

This is unit `EDialogConsole`

See also

- EBaseClasses Unit 53
- ECallStack Unit 54
- ECompatibility Unit 56
- ECore Unit 59
- EDialog Unit 64
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETypes Unit 118

7.34 EDialogHTMLBased

This is unit `EDialogHTMLBased`

See also

- EBaseClasses Unit 53
- ECallStack Unit 54
- ECompatibility Unit 56
- EDebugInfo Unit 62
- EDialog Unit 64
- EEncoding Unit 72
- EException Unit 81
- EInfoFormat Unit 89
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.35 EDialogISAPI

This is unit `EDialogISAPI`

See also

- EBaseClasses Unit 53

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ECompatibility Unit 56
- EDialog Unit 64
- EDialogWeb Unit 67
- EException Unit 81
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.36 EDialogSend

This is unit `EDialogSend`

See also

- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ELowLevel Unit 93
- ETypes Unit 118

7.37 EDialogSendWinAPI

This is unit `EDialogSendWinAPI`

See also

- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- EDialogSend Unit 67
- EListView Unit 90
- ELowLevel Unit 93
- EMonitors Unit 101
- ESysInfo Unit 115
- ETypes Unit 118

7.38 EDialogService

This is unit `EDialogService`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDialog Unit 64
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.39 EDialogWeb

This is unit `EDialogWeb`

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- ECallStack Unit 54

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ECompatibility Unit 56
- EDebugInfo Unit 62
- EDialog Unit 64
- EDialogHTMLBased Unit 66
- EEncoding Unit 72
- EException Unit 81
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.40 EDialogWER

This is unit `EDialogWER`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDialog Unit 64
- EException Unit 81
- EExceptionHook Unit 82
- EExceptionManager Unit 84
- EInfoFormat Unit 89
- ELogBuilder Unit 90
- ELowLevel Unit 93
- EModules Unit 101
- EPChars Unit 102
- ESysInfo Unit 115
- ETypes Unit 118
- EWER Unit 120

7.41 EDialogWinAPI

This is unit `EDialogWinAPI`

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- ECompatibility Unit 56
- EConsts Unit 59
- EDialog Unit 64
- EDLLs Unit 71
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.42 EDialogWinAPIEurekaLog

This is unit `EDialogWinAPIEurekaLog`

See also

- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDialog Unit 64
- EDialogWinApi Unit 68
- EEvents Unit 75
- EException Unit 81
- EListView Unit 90
- ELogBuilder Unit 90
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EMonitors Unit 101
- EResLeaks Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.43 EDialogWinAPIEurekaLogDetailed

This is unit `EDialogWinAPIEurekaLogDetailed`

See also

- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDialog Unit 64
- EDialogWinAPI Unit 68
- EDialogWinAPIEurekaLog Unit 69
- EDLLs Unit 71
- EException Unit 81
- EInfoFormat Unit 89
- EListView Unit 90
- ELogBuilder Unit 90
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EMonitors Unit 101
- ENT Unit 101
- EResLeaks Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.44 EDialogWinAPIMSClassic

This is unit `EDialogWinAPIMSClassic`

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDialog Unit 64
- EDialogWinApi Unit 68
- EEvents Unit 75
- EListView Unit 90
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EMonitors Unit 101
- EResLeaks Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.45 EDialogWinAPIStepsToReproduce

This is unit `EDialogWinAPIStepsToReproduce`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDialog Unit 64
- EDialogWinApi Unit 68
- EListView Unit 90
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EMonitors Unit 101
- EResLeaks Unit 102
- ETypes Unit 118

7.46 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 71
- EDisAsm Routines 70
- ETypes Unit 118

7.46.1 Routines

DecodeInstruction 287	Decodes a single CPU instruction.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

DisasmInstruction 290	Disassebles a single CPU instruction.
GetAsmSize 312	Returns size of a single machine opcode.
GetFunctionSize 326	Returns size of the function in bytes.
InstructionToString 341	Converts decoded CPU instruction to string represenstation.

See also

- EDisAsm Unit 70

7.46.2 Types

PInstruction 480	A pointer to TInstruction 537.
TInstruction 537	A CPU instruction.

See also

- EDisAsm Unit 70

7.47 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. DLL will be loaded and unloaded automatically.

See also

- ETypes Unit 118

7.48 EDNS

DNS support unit.

Remarks

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

See also

- EDNS Classes 72
- EDNS Routines 72
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- EStrConsts Unit 114
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.48.1 Classes

EDNSError 130	Base class for DNS errors.

See also

- EDNS Unit 71

7.48.2 Routines

GetMXServerFromEmail 328	Retrieves mail server for e-mail address.

See also

- EDNS Unit 71

7.49 EEncoding

Encoding support unit.

Remarks

This unit contains routines data encoding.

Note

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 73 unit for encryption, EHash 87 unit for hasing, ESignature 112 unit for reading certificate info from executables.

See also

- TEAEncrypt Function 447
- RSAEncrypt Function 414
- RSASign Function 422
- MemProtect Function 383
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 93
- EMLang Unit 100
- EPChars Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118

7.50 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 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 72 unit for encoding, EHash 87 unit for hasing, ESignature 112 unit for reading certificate info from executables.

See also

- TEAEncrypt Function 447
- TwofishEncrypt Function 457
- RSAEncrypt Function 414
- RSASign Function 422
- MemProtect Function 383
- EEncrypt Types 75
- EEncrypt Routines 73
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EEncoding Unit 72
- EHash Unit 87
- ELowLevel Unit 93
- ETypes Unit 118
- EWinCrypt Unit 121

7.50.1 Routines

EncryptBufferFree 295	Releases the encryption buffer.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

InitSalt 338	Generates random salt.
MemProtect 383	Encrypts process memory.
MemUnprotect 384	Decrypts process memory.
RSADecrypt 411	Decrypts encrypted data by using RSA cipher.
RSADestroyKey 414	Deletes RSA key.
RSAEncrypt 414	Encrypts "clear text" data by using RSA cipher.
RSAGenKey 417	Creates new RSA key.
RSALoadPrivateKey 418	Imports private key.
RSALoadPublicKey 419	Imports public key.
RSASavePrivateKey 420	Exports private key.
RSASavePublicKey 421	Exports public key.
RSASign 422	Signs data.
RSAVerify 424	Verifies digital signature.
SecureFree 428	Wipes and releases memory.
TEADecrypt 437	Decrypts encrypted data by using TEA cipher.
TEADefaultKey 441	Returns pre-initialized default key for TEA.
TEADeriveKey 443	Initializes TEA key from password.
TEADestroyKey 447	Fills TEA key with zeros.
TEAEncrypt 447	Encrypts "clear text" data by using TEA cipher.
TEAInitSessionKey 451	Generates random TEA key.
TwofishDecrypt 454	Decrypts encrypted data by using Twofish cipher.
TwofishDeriveKey 455	Initializes Twofish key from password.
TwofishDestroyKey 457	Deletes Twofish key.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TwofishEncrypt 457	Encrypts "clear text" data by using Twofish cipher.
TwofishInitIV 459	Creates new IV.
TwofishInitKey 459	"Imports" (precalculates) Twofish key from RAW key value.

See also

- EEncrypt Unit 73

7.50.2 Types

PSalt 481	Pointer to salt.
PTwofishInitVector 481	Pointer to IV for Twofish.
TEncryptBuffer 525	Contains an arbitrary array of bytes.
TRSAExport 551	Defines how to export/import RSA key.
TRSAKey 552	Key for RSA encryption.
TRSASignature 554	Digital signature.
TSalt 554	Salt value for deriving key from password.
TTEAKey 567	Key for TEA encryption.
TTwofishInitVector 569	Initialization Vector (IV) for Twofish.
TTwofishKey 569	(Precalculated) Key for Twofish encryption.
TTwofishRAWKey 570	(RAW) Key for Twofish encryption.

See also

- EEncrypt Unit 73

7.51 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

Instead of using events from code - you can subscribe to event by using EurekaLog component 29. 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;
...
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
type
   TForm1 = class
   ...
      procedure MyHandler(AExceptionInfo: TEurekaExceptionInfo; var AHar
   end;
implementation
...
procedure TForm1.FormCreate(Sender: TObject);
begin
   RegisterEventExceptionNotify(MyHandler);
   ...
end;
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 29
- Events 11
- EEvents Types 79
- EEvents Routines 77
- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EException Unit 81
- ELowLevel Unit 93
- ETypes Unit 118

7.51.1 Routines

RegisterEventAttachedFilesRequest 392	Registers OnAttachedFilesRequest event handler.
RegisterEventBeginReportGen 393	Registers OnBeginReportGen event handler.
RegisterEventCustomBugID 394	Registers OnCustomBugID event handler.
RegisterEventCustomButtonClick 395	Registers OnCustomButtonClick event handler.
RegisterEventCustomDataRequest 396	Registers OnCustomDataRequest event handler.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

RegisterEventCustomFileName 397	Registers OnCustomFileName event handler.
RegisterEventCustomWebFieldsRequest 398	Registers OnCustomWebFieldRequest event handler.
RegisterEventEndReportGen 399	Registers OnEndReportGen event handler.
RegisterEventExceptionAction 400	Registers OnExceptionAction event handler.
RegisterEventExceptionError 401	Registers OnExceptionError event handler.
RegisterEventExceptionNotify 402	Registers OnExceptionNotify event handler.
RegisterEventHelpButtonClick 403	Registers OnHelpButtonClick event handler.
RegisterEventPasswordRequest 404	Registers OnPasswordRequest event handler.
RegisterEventRaise 405	Registers OnRaise event handler.
RegisterEventReproduceRequest 406	Registers OnReproduceRequest event handler.
RegisterEventZippedFilesRequest 407	Registers OnZippedFilesRequest event handler.
UnRegisterEventAttachedFilesRequest 461	Unregisters previously registered OnAttachedFilesRequest event handler.
UnRegisterEventBeginReportGen 462	Unregisters previously registered OnBeginReportGen event handler.
UnRegisterEventCustomBugID 462	Unregisters previously registered OnCustomBugID event handler.
UnRegisterEventCustomButtonClick 463	Unregisters previously registered OnCustomButtonClick event handler.
UnRegisterEventCustomDataRequest 464	Unregisters previously registered OnCustomDataRequest event handler.
UnRegisterEventCustomFileName 465	Unregisters previously registered OnCustomFileName event handler.
UnRegisterEventCustomWebFieldsRequest 466	Unregisters previously registered OnCustomWebFieldRequest event

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	handler.
UnRegisterEventEndReportGen 466	Unregisters previously registered OnEndReportGen event handler.
UnRegisterEventExceptionAction 467	Unregisters previously registered OnExceptionAction event handler.
UnRegisterEventExceptionError 468	Unregisters previously registered OnExceptionError event handler.
UnRegisterEventExceptionNotify 469	Unregisters previously registered OnExceptionNotify event handler.
UnRegisterEventHelpButtonClick 470	Unregisters previously registered OnHelpButtonClick event handler.
UnRegisterEventPasswordRequest 470	Unregisters previously registered OnPasswordRequest event handler.
UnRegisterEventRaise 471	Unregisters previously registered OnRaise event handler.
UnRegisterEventReproduceRequest 472	Unregisters previously registered OnReproduceRequest event handler.
UnRegisterEventZippedFilesRequest 473	Unregisters previously registered OnZippedFilesRequest event handler.

See also

- EEvents Unit 75

7.51.2 Types

TELEvAttachedFilesRequestMeth 19	Method event handler for OnAttachedFilesRequest event.
TELEvAttachedFilesRequestProc 495	Procedural event handler for OnAttachedFilesRequest event.
TELEvCustomBugIDMeth 22	Method event handler for OnCustomBugID event.
TELEvCustomBugIDProc 496	Procedural event handler for OnCustomBugID event.
TELEvCustomButtonClickMeth 24	Method event handler for OnCustomButtonClick event.
TELEvCustomButtonClickProc 498	Procedural event handler for

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	OnCustomButtonClick event.
TELEvCustomDataRequestMeth 23	Method event handler for OnCustomDataRequest event.
TELEvCustomDataRequestProc 500	Procedural event handler for OnCustomDataRequest event.
TELEvCustomFileNameMeth 21	Method event handler for OnCustomFileName event.
TELEvCustomFileNameProc 501	Procedural event handler for OnCustomFileName event.
TELEvCustomWebFieldsRequestMeth 23	Method event handler for OnCustomWebFieldsRequest event.
TELEvCustomWebFieldsRequestProc 503	Procedural event handler for OnCustomWebFieldsRequest event.
TELEvExceptionActionMeth 15	Method event handler for OnExceptionAction event.
TELEvExceptionActionProc 505	Procedural event handler for OnExceptionAction event.
TELEvExceptionErrorMeth 17	Method event handler for OnExceptionError event.
TELEvExceptionErrorProc 508	Procedural event handler for OnExceptionError event.
TELEvExceptionNotifyMeth 14	Method event handler for OnExceptionNotify event.
TELEvExceptionNotifyProc 511	Procedural event handler for OnExceptionNotify event.
TELEvHelpButtonClickMeth 25	Method event handler for OnHelpButtonClick event.
TELEvHelpButtonClickProc 514	Procedural event handler for OnHelpButtonClick event.
TELEvPasswordRequestMeth 18	Method event handler for OnPasswordRequest event.
TELEvPasswordRequestProc 516	Procedural event handler for OnPasswordRequest event.
TELEvRaiseMeth 12	Method event handler for OnRaise event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TELEvRaiseProc 518	Procedural event handler for OnRaise event.
TELEvReportGenMeth 26	Method event handler for OnReportBeginGen/ OnReportEndGen events.
TELEvReportGenProc 522	Procedural event handler for OnReportBeginGen/ OnReportEndGen event.
TELEvReproduceRequestMeth 21	Method event handler for OnReproduceRequest event.
TELEvReproduceRequestProc 523	Procedural event handler for OnReproduceRequest event.
TELEvZippedFilesRequestMeth 20	Method event handler for OnZippedFilesRequest event.
TELEvZippedFilesRequestProc 525	Procedural event handler for OnZippedFilesRequest event.

See also

- EEvents Unit 75

7.52 EException

Unit with exception information class.

Examples

This unit holds TEurekaExceptionInfo 157 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 157
- ExceptionManager function 305
- EException Classes 82
- EAppType Unit 50
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EEvents Unit 75
- EExceptionInfo Unit 82
- EHash Unit 87
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETypes Unit 118

7.52.1 Classes

TEurekaExceptionInfo 157	Exception information class.

See also

- EException Unit 81

7.53 EExceptionHook

This is unit `EExceptionHook`

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EDLLs Unit 71
- EException Unit 81
- EExceptionInfo Unit 82
- EExceptionManager Unit 84
- EHook Unit 89
- EInject Unit 90
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- EThreadsManager Unit 117
- ETypes Unit 118

7.54 EExceptionInfo

This is unit `EExceptionInfo`

See also

- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EExceptionInfoDelphi2 Unit 83
- EExceptionInfoDelphiANSI Unit 83
- EExceptionInfoDelphiUnicode Unit 83
- EExceptionInfoGeneric Unit 83
- ELogging Unit 92

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

7.55 EExceptionInfoDelphi2

This is unit `EExceptionInfoDelphi2`

See also

- EBase Unit 52
- EClasses Unit 55
- EExceptionInfo Unit 82
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

7.56 EExceptionInfoDelphiANSI

This is unit `EExceptionInfoDelphiANSI`

See also

- EBase Unit 52
- EClasses Unit 55
- EExceptionInfo Unit 82
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

7.57 EExceptionInfoDelphiUnicode

This is unit `EExceptionInfoDelphiUnicode`

See also

- EBase Unit 52
- EClasses Unit 55
- EExceptionInfo Unit 82
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

7.58 EExceptionInfoGeneric

This is unit `EExceptionInfoGeneric`

See also

- EExceptionInfo Unit 82

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.59 EExceptionInfoMSVC

This is unit `EExceptionInfoMSVC`

7.60 EExceptionManager

See also

- ExceptionManager function 305
- TExceptionManager.Instance class function 204
- EExceptionManager Classes 84
- EExceptionManager Types 85
- EExceptionManager Routines 85
- EAppCrossModule Unit 49
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugEL Unit 60
- EDebugInfo Unit 62
- EDialog Unit 64
- EEvents Unit 75
- EException Unit 81
- EExceptionHook Unit 82
- EExceptionInfo Unit 82
- EFreeze Unit 86
- EHook Unit 89
- EInject Unit 90
- EInterfaces Unit 90
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- EStackTracing Unit 114
- EThreadsManager Unit 117
- ETypes Unit 118

7.60.1 Classes

TExceptionManager 203	Exception manager class.

See also

- EExceptionManager Unit 84

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.60.2 Routines

ExceptionManager 305	Exception manager.

See also

- EExceptionManager Unit 84

7.60.3 Types

TExceptionNotifyMode 534	Specifies work mode of exception manager.

See also

- EExceptionManager Unit 84

7.61 EFileMemory

This is unit `EFileMemory`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EHash Unit 87
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETools Unit 118
- ETypes Unit 118

7.62 EFixSafeCallException

This is unit `EFixSafeCallException`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EHook Unit 89
- ELowLevel Unit 93
- ETypes Unit 118

7.63 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EFonts Routines 86
- EBaseClasses Unit 53
- ETypes Unit 118

7.63.1 Routines

GUIFont 333	Returns font for GUI applications.
MonoFont 386	Returns fixed-width font.

See also

- EFonts Unit 85

7.64 EFreeze

Freeze detection unit.

Remarks

This unit holds code for running freeze detection checks.

Freeze detection is initialized automatically on startup by invoking InitCheckFreeze 337 function. This function uses FreezeThreadClass 585 variable to select specific hang detection code.

See also

- InitCheckFreeze Function 337
- FreezeThreadClass Variable 585
- EFreeze Classes 86
- EFreeze Types 87
- EFreeze Routines 87
- EFreeze Globals 87
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- ECompatibility Unit 56
- ECore Unit 59
- EDLLs Unit 71
- EException Unit 81
- EExceptionInfo Unit 82
- EExceptionManager Unit 84
- ELogging Unit 92
- ELowLevel Unit 93
- EModules Unit 101
- EStackTracing Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETypes Unit 118
- EWCT Unit 119
- EWCTSupport Unit 120

7.64.1 Classes

TFreezeThread 217	Base class for freeze detection threads.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TMainThreadFreezeDetectionThread 224	Thread class to detects hangs in UI thread.
TWCTFreezeDetectionThread 234	Thread class to detects deadlocks between threads.

See also
- EFreeze Unit 86

7.64.2 Globals

FreezeThreadClass 585	Determinate class to be used as background freeze detection thread.

See also
- EFreeze Unit 86

7.64.3 Routines

DoneCheckFreeze 291	Stops background freeze detection thread.
InitCheckFreeze 337	Starts checking for freezes and hangs.
PauseFreezeCheck 388	Temporarily disables freeze checks.
RaiseFreezeException 390	Raises freeze/hang/deadlock exception for the specified thread.
ResumeFreezeCheck 410	Enables freeze checks back.

See also
- EFreeze Unit 86

7.64.4 Types

TFreezeThreadClass 536	Class of freeze detection thread.

See also
- EFreeze Unit 86

7.65 EHash

Hashing support unit.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

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 73 unit for encryption, EEncoding 72 unit for encoding, ESignature 112 unit for reading certificate info from executables.

See also

- EHash Types 283
- EHash Routines 88
- EBaseClasses Unit 53
- ECompatibility Unit 56
- ELowLevel Unit 93
- ETypes Unit 118

7.65.1 Routines

CRC16Hash 283	Calculates CRC-16 checksum.
CRC32Hash 284	Calculates CRC-32 checksum.
MD5Hash 374	Calculates MD5 hash.
SDBMHash 427	Calculates SDBM hash.
SHA1Hash 434	Calculates SHA-1 hash.
SHA256Hash 434	Calculates SHA-2/SHA-256 hash.

See also

- EHash Unit 87

7.65.2 Types

TCRC16Hash 491	CRC-16 checksum.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TCRC32Hash 492	CRC-32 checksum.
TMD5Hash 540	MD5 hash.
TSDBMHash 555	SDBM hash.
TSHA1Hash 560	SHA-1 hash.
TSHA256Hash 560	SHA-2/SHA-256 hash.

See also

- EHash Unit 87

7.66 EHook

This is unit `EHook`

See also

- EAppType Unit 50
- EClassUtils Unit 56
- EDisAsm Unit 70
- ELogging Unit 92
- ELowLevel Unit 93
- ELowLevelClasses Unit 95
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ETypes Unit 118

7.67 EHTMLHelp

This is unit `EHTMLHelp`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71
- ELowLevel Unit 93
- ETypes Unit 118

7.68 EInfoFormat

This is unit `EInfoFormat`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.69 EInject

This is unit `EInject`

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

7.70 EInterfaces

This is unit `EInterfaces`

See also

- ETypes Unit 118

7.71 EJSON

This is unit `EJSON`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETypes Unit 118

7.72 EListView

This is unit `EListView`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- ELowLevel Unit 93
- ETypes Unit 118

7.73 ELogBuilder

Bug report builder unit.

Remarks

This unit holds declaration for abstract log builder class (TBaseLogBuilder 132) and 3 build-in specific implementation of log builder class (TLogBuilder 223, TXMLLogBuilder 235, and TDummyLogBuilder 139).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Log builder class is used by EurekaLog to create bug reports from exception information. Default class to use is defined by LogBuilderClass 586 global variable.

Note
Bug report is a simple multi-line text. Multi-files bug reports (ELP/ZIP) are created by dialog class 32

See also

- ELogBuilder Classes 91
- ELogBuilder Types 92
- ELogBuilder Routines 92
- ELogBuilder Globals 92
- EAppType Unit 50
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDebugInfo Unit 62
- EDisAsm Unit 70
- EEncoding Unit 72
- EEvents Unit 75
- EException Unit 81
- EHash Unit 87
- EHook Unit 89
- EInfoFormat Unit 89
- ELogging Unit 92
- ELogManager Unit 93
- ELowLevel Unit 93
- EModules Unit 101
- ESpecificDelphi Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETools Unit 118
- ETypes Unit 118
- EXMLBuilder Unit 123

7.73.1 Classes

TBaseLogBuilder 132	Base class for log builder.
TDummyLogBuilder 139	Dummy log builder class.
TLogBuilder 223	Default EurekaLog 7 log builder.
TXMLLogBuilder 235	XML EurekaLog 7+ log builder.

See also

- ELogBuilder Unit 90

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.73.2 Globals

LogBuilderClass 586	Specifies global log builder class to use.

See also
- ELogBuilder Unit 90

7.73.3 Routines

BuildBugReport 267	Builds bug report from exception information.
DisasmFunction 289	Disasms function.
FindLowAddress 308	Tool function to obtain starting address to disassemble code block around known address.

See also
- ELogBuilder Unit 90

7.73.4 Types

TELEvReportGenMeth 521	Method event handler for OnReportBeginGen/OnReportEndGen events.
TELEvReportGenProc 521	Procedural event handler for OnReportBeginGen/OnReportEndGen event.

See also
- ELogBuilder Unit 90

7.74 ELogging

This is unit `ELogging`

See also
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- EPChars Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.75 ELogManager

This is unit `ELogManager`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EEncrypt Unit 73
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESysInfo Unit 115
- ETypes Unit 118

7.76 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 93
- EConsts Unit 59
- EDLLs Unit 71
- ELogging Unit 92
- EPChars Unit 102
- ESpecificDelphi Unit 114
- ETypes Unit 118
- EWER Unit 120

7.76.1 Routines

AllocatePageAlign 260	No description available
CheckELMagicCode 268	No description available

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ConvertAddress 282	Convert address to code to real address of code.
CurrentInstance 286	Returns current instance.
GetReadableSize 329	Verifies that the caller has specified access to the specified range of memory.
IfElse 335	A replacement for ternary `if` operator.
IntoIDE 342	No description available
IsCompiledWithPackages 347	Checks if this module is compiled with packages.
IsDebugged 348	Determines whether the calling process is being debugged by a user-mode debugger.
IsReadableFile 360	Verifies that the caller has read access to the specified file.
IsRTLCompiledWithStackFrames 360	Determines whether the RTL was compiled with Stack Frames option enabled.
IsStackFramesProc 361	Determines whether the specified function has a stack frame.
IsValidBlockAddr 362	Verifies that the caller has read access to the specified range of memory.
IsValidClass 363	Verifies that the specified pointer is a valid class.
IsValidCodeAddr 364	Verifies that the caller has execute access to the specified range of memory.
IsValidHandle 365	Verifies that this is a valid kernel handle.
IsValidObject 365	Verifies that the specified pointer is a valid object.
IsWritableFile 373	Verifies that the caller has write access to the specified file.
MainThreadTopOfStack 374	Returns top of the stack for the main thread.
PageSize 388	No description available
PrimaryInstance 389	The same as MainInstance. See remarks

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	for more info.
SafeExec 426	Executes routine with exception handling. Any exception is handled as fatal exception.

See also

- ELowLevel Unit 93

7.77 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).

7.78 EMantisConnectStub

This is unit `EMantisConnectStub`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESOAPMantis Unit 113
- ETools Unit 118
- ETypes Unit 118
- EXML Unit 123
- EXMLRPC Unit 123

7.79 EMAPI

This is unit `EMAPI`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- ESysInfo Unit 115
- ETypes Unit 118

7.80 EMapParserBase

This is unit `EMapParserBase`

See also

- EMapParserBase Classes 96
- EBase Unit 52

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EFileMemory Unit 85
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EResLeaks Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118

7.80.1 Classes

TELAbstractMapParser 139	Base class for .map file parsers.

See also

- EMapParserBase Unit 95

7.81 EMapWin32

This is unit `EMapWin32`

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EHook Unit 89
- ELowLevel Unit 93
- ETypes Unit 118

7.82 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.

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 157 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 358/IsMemLeaksEnabled 358 functions;

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EnableMemLeaksRegistering 294/DisableMemLeaksRegistering 288 functions;
- EurekaRegisterExpectedMemoryLeak 300/EurekaUnregisterExpectedMemoryLeak 303 functions;
- AllocMemSizeEx 262/AllocMemCountEx 261 functions;
- AllocMemSizeMax 263/AllocMemCountMax 262 functions;
- MemLeaksName function 375;
- MemLeaksOwn function 377;
- MemLeaksSetParentBlock function 380;
- CheckHeap function 268;
- DumpAllocationsToFile function 292;
- EurekaTryGetMem function 303;

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.

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,

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 100
- EMemLeaks Routines 98
- EConsts Unit 59
- EDisAsm Unit 70
- EDLLs Unit 71
- ELogging Unit 92
- ELowLevel Unit 93
- EPChars Unit 102
- EStackTracing Unit 114
- ETypes Unit 118

7.82.1 Routines

AllocMemCountEx 261	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemCountMax 262	Represents the maximum number of allocated memory blocks.
AllocMemSizeEx 262	Represents the total size of allocated memory blocks.
AllocMemSizeMax 263	Represents the maximum size of allocated memory.
CheckHeap 268	Checks heap for consistency.
DisableMemLeaksRegistering 288	Disables memory leaks checks.
DumpAllocationsToFile 292	Logs all allocated memory to file.
EnableMemLeaksRegistering 294	Enables memory leaks checks.
EurekaAllocMem 297	AllocMem allocates a memory block and initializes each byte to zero.
EurekaFreeMem 298	Frees a memory block previously allocated with EurekaGetMem 298, EurekaTryGetMem 303 or EurekaAllocMem 297.
EurekaGetMem 298	Allocates a memory block via EurekaLog memory filter.
EurekaReallocMem 299	Resizes a memory block.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EurekaRegisterExpectedMemoryLeak 300	Register an expected memory leak.
EurekaRegisterExpectedMemoryLeakDynArray 301	Register an expected memory leak (variant for dynamic arrays).
EurekaRegisterExpectedMemoryLeakStr 302	Register an expected memory leak (variant for strings).
EurekaTryGetMem 303	Allocates a memory block via EurekaLog memory filter (w/o error checking).
EurekaUnregisterExpectedMemoryLeak 303	Removes memory block from excluded leaks.
EurekaUnregisterExpectedMemoryLeakDynArray 304	Removes memory block from excluded leaks (variant for dynamic arrays).
EurekaUnregisterExpectedMemoryLeakStr 305	Removes memory block from excluded leaks (variant for strings).
IsMemLeaksEnabled 358	Indicate if memory leaks checks are enabled.
IsMemLeaksInstalled 358	No description available
MemLeaksName 375	Names a memory block.
MemLeaksNameDynArray 376	Names a memory block (variant for dynamic arrays).
MemLeaksNameStr 376	Names a memory block (variant for strings).
MemLeaksOwn 377	Sets ownership for memory block.
MemLeaksOwnDynArrayBoth 378	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayChild 378	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayParent 379	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnStr 380	Sets ownership for memory block (variant for strings).
MemLeaksSetParentBlock 380	No description available
MemLeaksSetParentBlockDynArrayBoth 381	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

MemLeaksSetParentBlockDynArrayChild 381	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayParent 382	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockStr 382	Sets parent-child relation between two memory blocks for grouping (variant for strings).

See also

- EMemLeaks Unit 96

7.82.2 Types

PCustomName 480	Defines buffer for storing name of memory block.
PPointerRecord 480	A pointer to TPointerRecord type 545
TBlockType 483	Defines type of memory block.
TCustomName 492	Defines buffer for storing name of memory block.
TDumpCompareFunc 493	Comparator for memory blocks.
TDumpFilterFunc 493	Filter for memory blocks.
TMemoryError 540	Type of detected memory error.
TMemoryErrors 543	Defines set of possible memory errors.
TPointerRecord 545	Header for memory blocks allocated by EurekaLog.

See also

- EMemLeaks Unit 96

7.83 EMLang

MLang headers.

Remarks

This unit contains translation of mlang.h.

See also

- MLang

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.84 EModules

This is unit `EModules`

See also

- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- EHook Unit 89
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EPEImage Unit 102
- ESpecificDelphi Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118

7.85 EMonitors

This is unit `EMonitors`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- ETypes Unit 118

7.86 ENT

This is unit `ENT`

See also

- EDLLs Unit 71
- ELowLevel Unit 93
- EPEImage Unit 102
- ETypes Unit 118

7.87 EOSApiList

This is unit `EOSApiList`

See also

- EHook Unit 89
- ELowLevel Unit 93
- EResLeaks Unit 102
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.88 EPChars

This is unit EPChars

See also

- ETypes Unit 118

7.89 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

- EAnsiFileNamesFix Unit 48
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118
- EUnmangling Unit 119

7.90 EPNG

This is unit EPNG

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ELowLevel Unit 93
- ETypes Unit 118
- EZLib Unit 125

7.91 EResLeaks

This is unit EResLeaks

See also

- EResLeaks Types 103
- EResLeaks Routines 103
- EConsts Unit 59
- EDisAsm Unit 70
- EDLLs Unit 71
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- ELowLevelClasses Unit 95
- EMemLeaks Unit 96
- EOSApiList Unit 101
- EStackTracing Unit 114
- EStrConsts Unit 114
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.91.1 Routines

HookWin32API 334	Adds a function to resource monitoring.

See also
- EResLeaks Unit 102

7.91.2 Types

TCheckType 489	Function check kind.

See also
- EResLeaks Unit 102

7.92 ERunTime

This is unit `ERunTime`

See also
- EAnsiFileNamesFix Unit 48
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- EClassUtils Unit 56
- ECompatibility Unit 56
- EComponent Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDbgHelp Unit 60
- EDebugDWARF Unit 60
- EDebugEL Unit 60
- EDebugEL7 Unit 61
- EDebugExports Unit 61
- EDebugInfo Unit 62
- EDebugJCL Unit 62
- EDebugMap Unit 62
- EDebugME Unit 63
- EDebugMS Unit 63
- EDebugStabs Unit 63
- EDebugTD32 Unit 64
- EDialog Unit 64
- EDialogCGI Unit 66
- EDialogConsole Unit 66
- EDialogHTMLBased Unit 66
- EDialogISAPI Unit 66
- EDialogSend Unit 67
- EDialogSendWinAPI Unit 67
- EDialogService Unit 67
- EDialogWeb Unit 67
- EDialogWER Unit 68
- EDialogWinAPI Unit 68

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EDialogWinAPIEurekaLog Unit 69
- EDialogWinAPIEurekaLogDetailed Unit 69
- EDialogWinAPIMSClassic Unit 69
- EDialogWinAPIStepsToReproduce Unit 70
- EDisAsm Unit 70
- EDLLs Unit 71
- EDNS Unit 71
- EEncoding Unit 72
- EEncrypt Unit 73
- EEvents Unit 75
- EException Unit 81
- EExceptionHook Unit 82
- EExceptionInfo Unit 82
- EExceptionInfoDelphi2 Unit 83
- EExceptionInfoDelphiANSI Unit 83
- EExceptionInfoDelphiUnicode Unit 83
- EExceptionInfoGeneric Unit 83
- EExceptionInfoMSVC Unit 84
- EExceptionManager Unit 84
- EFileMemory Unit 85
- EFonts Unit 85
- EFreeze Unit 86
- EHash Unit 87
- EHook Unit 89
- EHTMLHelp Unit 89
- EInfoFormat Unit 89
- EInject Unit 90
- EInterfaces Unit 90
- EJSON Unit 90
- EListView Unit 90
- ELogBuilder Unit 90
- ELogging Unit 92
- ELogManager Unit 93
- ELowLevel Unit 93
- ELowLevelClasses Unit 95
- EMantisConnectStub Unit 95
- EMAPI Unit 95
- EMapParserBase Unit 95
- EMemLeaks Unit 96
- EMLang Unit 100
- EModules Unit 101
- EMonitors Unit 101
- ENT Unit 101
- EOSApiList Unit 101
- EPChars Unit 102
- EPEImage Unit 102
- EPNG Unit 102
- EResLeaks Unit 102
- EScreenShot Unit 105
- ESend Unit 105
- ESendAPIBugZilla Unit 106
- ESendAPIFogBugz Unit 106
- ESendAPIJIRA Unit 107
- ESendAPIMantis Unit 107
- ESendMail Unit 108
- ESendMailMAPI Unit 108
- ESendMailShell Unit 109
- ESendMailSMAPI Unit 109
- ESendMailSMTP Unit 110
- ESendWeb Unit 111
- ESendWebFTP Unit 111

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ESendWebHTTP Unit 111
- ESignature Unit 112
- ESOAP Unit 113
- ESOAPMantis Unit 113
- ESocket Unit 113
- ESockets Unit 113
- ESpecificDelphi Unit 114
- ESSLSocket Unit 114
- EStackTracing Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETaskDialogs Unit 117
- EThreadsManager Unit 117
- ETools Unit 118
- ETypes Unit 118
- EUnmangling Unit 119
- EWCT Unit 119
- EWCTSupport Unit 120
- EWebTools Unit 120
- EWER Unit 120
- EWinCrypt Unit 121
- EWinSock Unit 121
- ExceptionLog7 Unit 121
- EXML Unit 123
- EXMLBuilder Unit 123
- EXMLRPC Unit 123
- EXMLRPCBugZilla Unit 124
- EZip Unit 124
- EZLib Unit 125
- EZLibHeaders Unit 125

7.93 EScreenShot

This is unit `EScreenShot`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ELowLevel Unit 93
- EPNG Unit 102
- ESysInfo Unit 115
- ETypes Unit 118

7.94 ESend

This is unit `ESend`

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELogging Unit 92
- ELowLevel Unit 93

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EMemLeaks Unit 96
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETypes Unit 118

7.95 ESendAPIBugZilla

This is unit `ESendAPIBugZilla`

See also

- ESendAPIBugZilla Classes 106
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EHash Unit 87
- ELowLevel Unit 93
- ESend Unit 105
- ESendWeb Unit 111
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118
- EWebTools Unit 120
- EXMLRPC Unit 123
- EXMLRPCBugZilla Unit 124

7.95.1 Classes

TELTrackerBugZillaSender 147	Implements API access to BugZilla bug tracker

See also

- ESendAPIBugZilla Unit 106

7.96 ESendAPIFogBugz

This is unit `ESendAPIFogBugz`

See also

- ESendAPIFogBugz Classes 107
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EEncoding Unit 72
- ELowLevel Unit 93
- ESend Unit 105
- ESendWeb Unit 111
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETypes Unit 118
- EWebTools Unit 120
- EXMLBuilder Unit 123

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.96.1 Classes

TELTrackerFogBugzSender 41	Implements API access to Fogbugz bug tracker

See also
- ESendAPIFogBugz Unit 106

7.97 ESendAPIJIRA

This is unit ESendAPIJIRA

See also
- ESendAPIJIRA Classes 107
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EJSON Unit 90
- ELowLevel Unit 93
- ESend Unit 105
- ESendWeb Unit 111
- EStrConsts Unit 114
- ETypes Unit 118
- EWebTools Unit 120

7.97.1 Classes

TELTrackerJIRASender 148	Implements API access to JIRA bug tracker

See also
- ESendAPIJIRA Unit 107

7.98 ESendAPIMantis

This is unit ESendAPIMantis

See also
- ESendAPIMantis Classes 108
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EExceptionHook Unit 82
- ELowLevel Unit 93
- EMantisConnectStub Unit 95
- EModules Unit 101
- ESend Unit 105
- ESendWeb Unit 111
- ESpecificDelphi Unit 114

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118
- EWebTools Unit 120
- EXML Unit 123
- EXMLRPC Unit 123

7.98.1 Classes

TELTrackerMantisSender 40	Implements API access to Mantis bug tracker

See also

- ESendAPIMantis Unit 107

7.99 ESendMail

This is unit `ESendMail`

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESend Unit 105
- ESendMailShell Unit 109
- ESysInfo Unit 115
- ETypes Unit 118

7.100 ESendMailMAPI

This is unit `ESendMailMAPI`

See also

- ESendMailMAPI Classes 109
- EAppType Unit 50
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- EMAPI Unit 95
- ESend Unit 105
- ESendMail Unit 108
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 118

7.100.1 Classes

TELMailMAPISender 38	Implements sending via full MAPI protocol.

See also

- ESendMailMAPI Unit 108

7.101 ESendMailShell

This is unit `ESendMailShell`

See also

- ESendMailShell Classes 109
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- ELowLevel Unit 93
- ESend Unit 105
- ESendMail Unit 108
- ESysInfo Unit 115
- ETypes Unit 118

7.101.1 Classes

TELMailShellSender 37	Implements sending via mailto: protocol

See also

- ESendMailShell Unit 109

7.102 ESendMailSMAPI

This is unit `ESendMailSMAPI`

See also

- ESendMailSMAPI Classes 110
- EAppType Unit 50
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- EMAPI Unit 95
- ESend Unit 105

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ESendMail Unit 108
- EStrConsts Unit 114
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118

7.102.1 Classes

TELMailSimpleMAPISender 37	Implements sending via Simple MAPI

See also

- ESendMailSMAPI Unit 109

7.103 ESendMailSMTP

This is unit `ESendMailSMTP`

See also

- ESendMailSMTP Classes 110
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EDNS Unit 71
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESend Unit 105
- ESendMail Unit 108
- ESocket Unit 113
- ESockets Unit 113
- ESysInfo Unit 115
- ETools Unit 118
- ETypes Unit 118
- EWinSock Unit 121

7.103.1 Classes

TELMailSMTPBaseSender 146	Base class for SMTP sending.
TELMailSMTPClientSender 39	Implements sending via SMTP client protocol.
TELMailSMTPSender 38	No description available
TELMailSMTPServerSender 38	Implements sending via SMTP server protocol.

See also

- ESendMailSMTP Unit 110

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.104 ESendWeb

This is unit ESendWeb

See also

- EBaseClasses Unit 53
- EClasses Unit 55
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- EEvents Unit 75
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ESend Unit 105
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EWebTools Unit 120

7.105 ESendWebFTP

This is unit ESendWebFTP

See also

- ESendWebFTP Classes 111
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- ECore Unit 59
- EEncoding Unit 72
- ELowLevel Unit 93
- ESend Unit 105
- ESendWeb Unit 111
- ETypes Unit 118
- EWebTools Unit 120

7.105.1 Classes

TELFTPSender 39	Implements sending via FTP uploading.

See also

- ESendWebFTP Unit 111

7.106 ESendWebHTTP

This is unit ESendWebHTTP

See also

- ESendWebHTTP Classes 112
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EEncoding Unit 72
- ELowLevel Unit 93

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EMemLeaks Unit 96
- EResLeaks Unit 102
- ESend Unit 105
- ESendWeb Unit 111
- ETypes Unit 118
- EWebTools Unit 120

7.106.1 Classes

TELHTTPSender 39	Implements sending via HTTP posting.

See also

- ESendWebHTTP Unit 111

7.107 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 73 unit.

See also

- SignatureGetInfo Function 436
- EEncrypt Unit 73
- EEncoding Unit 72
- EHash Unit 87
- ESignature Types 112
- ESignature Routines 112
- EBaseClasses Unit 53
- ECompatibility Unit 56
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118
- EWinCrypt Unit 121

7.107.1 Routines

SignatureGetInfo 436	Returns information about digital signature and its certificate of an executable file.

See also

- ESignature Unit 112

7.107.2 Types

TCertificateInfo 485	Certificate information.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TSignatureInfo 561	Information about digital signature of executable file.

See also

- ESignature Unit 112

7.108 ESOAP

This is unit ESOAP

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETypes Unit 118

7.109 ESOAPMantis

This is unit ESOAPMantis

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118
- EXML Unit 123
- EXMLRPC Unit 123

7.110 ESocket

This is unit ESocket

See also

- ELowLevel Unit 93
- ETypes Unit 118
- EWinSock Unit 121

7.111 ESockets

This is unit ESockets

See also

- EAppType Unit 50
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EDNS Unit 71

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EEncoding Unit 72
- ELogging Unit 92
- ELowLevel Unit 93
- ESocket Unit 113
- ESSLSocket Unit 114
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118
- EWinCrypt Unit 121
- EWinSock Unit 121

7.112 ESpecificDelphi

This is unit `ESpecificDelphi`

See also

- ELowLevel Unit 93
- ETypes Unit 118

7.113 ESSLSocket

This is unit `ESSLSocket`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71
- ELowLevel Unit 93
- ESocket Unit 113
- ETypes Unit 118
- EWinCrypt Unit 121
- EWinSock Unit 121

7.114 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 54. Do not use code from this unit directly - always use `ECallStack` unit.

See also

- ECallStack unit 54
- EConsts Unit 59
- EDisAsm Unit 70
- EDLLs Unit 71
- ELowLevel Unit 93
- ETypes Unit 118

7.115 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

localization solution may use any approach. For example, it may localize resourcestrings. Or it may install OnTranslate handler.

See also

- EConsts Unit 59
- ELowLevel Unit 93
- EMemLeaks Unit 96

7.116 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 89
- Log builder 90
- ESysInfo Routines 115
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- ENT Unit 101
- EResLeaks Unit 102
- EStrConsts Unit 114
- ETools Unit 118
- ETypes Unit 118

7.116.1 Routines

ExpandEnvVars 306	Expands environment-variable strings.
GetActiveFormClass 308	Gets window class of active window.
GetActiveFormText 309	Gets caption of active window.
GetApplicationDescription 310	Gets application description.
GetApplicationName 310	Gets application name.
GetApplicationRect 311	Gets bounding rectangle for GUI application.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

GetCompanyName 313	Gets company name.
GetComputerName 314	Gets computer name user name.
GetControlClass 314	Gets class of active control.
GetControlText 315	Gets text of active control.
GetCurrentModuleCompilationDate 316	Gets compilation time of current module.
GetCurrentUserPrivileges 316	Gets description of user priviledges.
GetDiskInfo 317	Gets total and free disk space size.
GetDisplayDPI 317	Gets screen DPI.
GetDisplayMode 318	Gets screen mode.
GetEnvVariable 318	Expands single environment variable.
GetEurekaLogLabel 319	Gets full EurekaLog version.
GetEurekaLogLicense 320	Gets EurekaLog license.
GetEurekaLogPath 321	Obsolete function, do not use.
GetEurekaLogStatus 322	Gets EurekaLog stable/beta information.
GetEurekaLogVersion 322	Gets EurekaLog version.
GetExceptionModuleName 323	No description available
GetExceptionModuleVersion 324	No description available
GetFileIcon 324	No description available
GetFileSize 325	Gets file size.
IsWin10 368	Detect Windows 10 (minimum).
IsWin64 368	Detect OS bitness.
IsWin7 369	Detect Windows 7 (minimum).
IsWin8 369	Detect Windows 8 (minimum).
IsWin81 370	Detect Windows 8.1 (minimum).
IsWinVista 371	Detect Windows Vista (minimum).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

IsWinXP 371	Detect Windows XP (minimum).
IsWorkstation 372	Gets server/workstation status of current computer.

See also

- ESysInfo Unit 115

7.117 ETaskDialogs

This is unit `ETaskDialogs`

See also

- ETypes Unit 118

7.118 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 190
- ThreadData Function 452
- ThreadDataEnum Function 453
- EThreadsManager Classes 117
- EThreadsManager Types 118
- EThreadsManager Routines 118
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EHook Unit 89
- ELogging Unit 92
- ELowLevel Unit 93
- EMemLeaks Unit 96
- ENT Unit 101
- EResLeaks Unit 102
- ETypes Unit 118

7.118.1 Classes

TEurekaThreadData 190	Thread information.
TEurekaThreadDataList 200	List of threads.

See also

- EThreadsManager Unit 117

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

7.118.2 Routines

AquireThreadData 264	Returns thread information (any thread).
NameThread 386	Alias for NameThreadForDebugging to avoid conflicts with System.Classes.TThread.NameThreadForDebugging.
NameThreadForDebugging 387	Sets the name for a thread.
ThreadData 452	Returns thread information (current thread only).
ThreadDataEnum 453	Enumerates running threads.

See also

- EThreadsManager Unit 117

7.118.3 Types

TThreadKind 568	Indicates thread kind.

See also

- EThreadsManager Unit 117

7.119 ETools

This is unit `ETools`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- EDLLs Unit 71
- ELowLevel Unit 93
- EStrConsts Unit 114
- ETypes Unit 118

7.120 ETypes

Basic EurekaLog types.

Remarks

This unit holds declarations of basic types and constants.

Tip

This unit is also `interface` counterpart to ECompatibility 56 unit.

That's why this unit must be listed as last unit in `uses` clause of `interface` section.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ECompatibility 56
- ETypes Types 119

7.120.1 Types

TAppType 482	Defines application type.
TBugID 484	BugID type.
TEurekaActionType 526	Action kind for exception processing.
TEurekaLogFileType 531	Type of file name.
TEurekaModuleType 533	Alias for TModuleType 543.
TModuleType 543	Indicates kind of executable module.
TResponse 547	No description available
TSendResult 555	Result kind for exception processing.

See also

- ETypes Unit 118

7.121 EUnmangling

This is unit `EUnmangling`

See also

- EBase Unit 52
- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71
- ELowLevel Unit 93
- EPEImage Unit 102
- ETools Unit 118
- ETypes Unit 118

7.122 EWCT

WCT headers.

Remarks

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

Call WCTInit 474 function before using this unit. (Optionally) Call WCTDone 474 function after using this unit.

See also

- EWCT Routines 120
- EDLLs Unit 71

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ELowLevel Unit 93

7.122.1 Routines

WCTDone 474	Finalize EWCT unit.
WCTInit 474	Initialize EWCT unit.

See also

- EWCT Unit 119

7.123 EWCTSupport

This is unit `EWCTSupport`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EDLLs Unit 71
- ETypes Unit 118
- EWCT Unit 119

7.124 EWebTools

This is unit `EWebTools`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EConsts Unit 59
- EDLLs Unit 71
- EEncoding Unit 72
- ELowLevel Unit 93
- ESend Unit 105
- EStrConsts Unit 114
- ETypes Unit 118

7.125 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 340 function before using this unit.

See also

- InitWER 340
- ReportFault
- WerReportSubmit
- EWER Routines 121
- EDLLs Unit 71
- ELowLevel Unit 93

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 118

7.125.1 Routines

InitWER 340	Initializes EWER unit.

See also

- EWER Unit 120

7.126 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.

7.127 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 71
- ELowLevel Unit 93

7.128 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 52 unit if you want to call EurekaLog routines in application without EurekaLog enabled (such as IsEurekaLogActive 349). 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 code.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 122
- EAnsiFileNamesFix Unit 48
- EAppType Unit 50
- EBase Unit 52
- EBaseClasses Unit 53
- ECallStack Unit 54
- EClasses Unit 55
- EClassUtils Unit 56
- ECompatibility Unit 56
- EConfig Unit 57
- EConsts Unit 59
- ECore Unit 59
- EDebugEL Unit 60
- EDebugInfo Unit 62
- EDialog Unit 64
- EDisAsm Unit 70
- EDLLs Unit 71
- EEncoding Unit 72
- EEncrypt Unit 73
- EException Unit 81
- EExceptionHook Unit 82
- EExceptionInfo Unit 82
- EExceptionManager Unit 84
- EFreeze Unit 86
- EHook Unit 89
- EInfoFormat Unit 89
- EInject Unit 90
- ELogging Unit 92
- ELogManager Unit 93
- ELowLevel Unit 93
- EMemLeaks Unit 96
- EModules Unit 101
- ENT Unit 101
- EPEImage Unit 102
- EResLeaks Unit 102
- ESend Unit 105
- ESendMail Unit 108
- ESendMailShell Unit 109
- ESpecificDelphi Unit 114
- EStackTracing Unit 114
- EStrConsts Unit 114
- ESysInfo Unit 115
- EThreadsManager Unit 117
- ETypes Unit 118

7.128.1 Routines

CurrentEurekaLogOptions 8	No description available
IsEurekaLogActive 3	Indicates if EurekaLog is active or disabled in this application.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 121

7.129 EXML

This is unit EXML

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118

7.130 EXMLBuilder

This is unit EXMLBuilder

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118

7.131 EXMLRPC

This is unit EXMLRPC

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- EEncoding Unit 72
- ELowLevel Unit 93
- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EWebTools Unit 120
- EXML Unit 123

7.132 EXMLRPCBugZilla

This is unit `EXMLRPCBugZilla`

See also

- EBaseClasses Unit 53
- ECompatibility Unit 56
- ECore Unit 59
- EEncoding Unit 72
- EInfoFormat Unit 89
- ELowLevel Unit 93
- ETools Unit 118
- ETypes Unit 118
- EXML Unit 123
- EXMLRPC Unit 123

7.133 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 125
- EZip Classes 124
- EZip Types 125
- EZip Routines 125
- EZip Globals 124
- EBaseClasses Unit 53
- ETypes Unit 118

7.133.1 Classes

EZIPException 131	Exception type for all Zip errors.
TZipFile 235	Class for creating and reading .ZIP files.

See also

- EZip Unit 124

7.133.2 Globals

SIGNATURE_CENTRALHEADER 586	Signature of .zip file.
SIGNATURE_DATADESCRIPTOR 587	Data descriptor signature.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

SIGNATURE_LOCALHEADER 587	Local file header signature.
SIGNATURE_ZIPENDOFHEADER 588	End of central dir signature.

See also
- EZip Unit 124

7.133.3 Routines

TZipCompressionToString 460	Converts ZIP compression method value to string representation.

See also
- EZip Unit 124

7.133.4 Types

TStreamConstructor 567	Function to Create a Compression/ Decompression stream
TZipCompression 570	Zip compression method.
TZipEndOfCentralHeader 576	Final block written to zip file
TZipHeader 577	TZipHeader contains information about a file in a zip archive.

See also
- EZip Unit 124

7.134 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 124

7.135 EZLibHeaders

This is unit `EZLibHeaders`

See also
- EBaseClasses Unit 53
- ECompatibility Unit 56

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ETypes Unit 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part VII

8 All Classes

EDNSError 130	Base class for DNS errors.
EEurekaLogGeneralError 131	No description available
EZIPException 131	Exception type for all Zip errors.
TBaseDialog 32	Base class for all dialogs.
TBaseLogBuilder 132	Base class for log builder.
TConsoleWriterDialog 33	No description available
TDummyLogBuilder 139	Dummy log builder class.
TELAbstractMapParser 139	Base class for .map file parsers.
TELDebugInfoExports 35	No description available
TELDebugInfoJedi 35	No description available
TELDebugInfoMadExcept 35	No description available
TELDebugInfoMap 35	No description available
TELDebugInfoNative 35	No description available
TELDebugInfoSymbols 35	No description available
TELDebugInfoTD32 35	No description available
TELFTPSender 39	Implements sending via FTP uploading.
TELHTTPSender 39	Implements sending via HTTP posting.
TELMailMAPISender 38	Implements sending via full MAPI protocol.
TELMailShellSender 37	Implements sending via mailto: protocol
TELMailSimpleMAPISender 37	Implements sending via Simple MAPI
TELMailSMTPBaseSender 146	Base class for SMTP sending.
TELMailSMTPClientSender 39	Implements sending via SMTP client protocol.
TELMailSMTPSender 38	No description available

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TELMailSMTPServerSender 38	Implements sending via SMTP server protocol.
TELTrackerBugZillaSender 40	No description available
TELTrackerBugZillaSender 147	Implements API access to BugZilla bug tracker
TELTrackerFogBugzSender 41	Implements API access to Fogbugz bug tracker
TELTrackerFogBugzSender 40	No description available
TELTrackerJIRASender 148	Implements API access to JIRA bug tracker
TELTrackerMantisSender 40	Implements API access to Mantis bug tracker
TELTrackerMantisSender 40	No description available
TEurekaBaseStackList 149	Base call stack class.
TEurekaExceptionInfo 157	Exception information class.
TEurekaLogDetailedDialog 33	No description available
TEurekaLogDialog 33	No description available
TEurekaLogEvents 29	EurekaLog 7 component.
TEurekaModuleOptions 188	EurekaLog options.
TEurekaStackListV6 188	Stack tracing class for EurekaLog 6 method.
TEurekaStackListV7 189	Stack tracing class for EurekaLog 7 RAW method.
TEurekaThreadData 190	Thread information.
TEurekaThreadDataList 200	List of threads.
TEventLogDialog 33	No description available
TExceptionManager 203	Exception manager class.
TFreezeThread 217	Base class for freeze detection threads.
TGenericStackList 220	Common stack tracing class.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TLogBuilder 223	Default EurekaLog 7 log builder.
TMainThreadFreezeDetectionThread 224	Thread class to detects hangs in UI thread.
TMessageBoxDetailedDialog 33	No description available
TMessageBoxDialog 33	No description available
TMSClassicDialog 33	No description available
TNullDialog 33	No description available
TRTLHandlerDialog 33	No description available
TThreadEx 225	Abstract class to create separate threads of execution.
TWCTFreezeDetectionThread 234	Thread class to detects deadlocks between threads.
TWebDialog 33	No description available
TWebDialog 33	No description available
TWebDialog 33	No description available
TWERDialog 33	No description available
TXMLLogBuilder 235	XML EurekaLog 7+ log builder.
TZipFile 235	Class for creating and reading .ZIP files.

8.1 EDNSError

Base class for DNS errors.

Unit

EDNS 71

Syntax

Code (Delphi)

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

Remarks

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- GetMXServerFromEmail 328

8.2 EEurekaLogGeneralError

Base exception class for EurekaLog's exceptions.

Unit

EBase 52

Syntax

Code (Delphi)

```
EEurekaLogGeneralError = class(Exception);
```

Remarks

You can use this exception class in exception filters or exception events to filter out EurekaLog-related exceptions.

8.3 EZIPException

Exception type for all Zip errors.

Unit

EZip 124

Syntax

Code (Delphi)

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

Remarks

8.4 TBaseDialog

Base class for all dialogs.

Unit

EDialog 64

Syntax

Code (Delphi)

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

Remarks

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

common interface.

Actual (non abstract) dialogs are implemented as child sub-classes for this class. Each dialog class must be registered via RegisterDialogClass 391 or RegisterDialogClassFirst 391 functions.

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

8.5 TBaseLogBuilder

Base class for log builder.

Unit

ELogBuilder 90

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 223.

Another build-in log builder classes are TXMLLogBuilder 235 and TDummyLogBuilder 139.

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 267.

Note

Bug report is a simple multi-line text. Multi-files bug reports (ELP/ZIP) are created by dialog class 32.

Methods 132	Methods
Properties 135	Properties
protected	Protected members
public	Public members

8.5.1 Methods

Create 133	Creates instance of log builder class.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

BuildReport 133	Builds bug report.
Refresh 134	Refreshes (i.e. recreates) bug report.
ToString 135	Returns textual representation of class instance.

See also

- TBaseLogBuilder 132
- ELogBuilder 90

8.5.1.1 Create

Creates instance of log builder class.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
public
constructor Create(
   const AExceptionInfo: TEurekaExceptionInfo 157
); virtual;
```

Parameters

AExceptionInfo [in]

Exception information to build bug report. This information is saved to TBaseLogBuilder.ExceptionInfo 137 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 138 property is accessed. I.e. bug report is created on demand.

See also

- CustomFields property 137
- ExceptionInfo property 137
- Options property 138
- Report property 138
- BuildReport method 133
- ToString method 135
- TBaseLogBuilder Class 132

8.5.1.2 BuildReport

Builds bug report.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
ELogBuilder 90

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 137 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 135 method or TBaseLogBuilder.Report 138 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 134

See also
- Refresh method 134
- FBuilded field 136
- Options property 138
- CallStack property 136
- ExceptionInfo property 137
- CustomFields property 137
- ToString method 135
- ESysInfo unit 115
- TBaseLogBuilder Class 132

8.5.1.3 Refresh

Refreshes (i.e. recreates) bug report.

Unit
ELogBuilder 90

Syntax

Code (Delphi)
```
protected
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
procedure Refresh;
   virtual;
```

Remarks

Sets TBaseLogBuilder.FBuilded 136 to False, so next access to TBaseLogBuilder.Report 138 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 136
- BuildReport method 133
- Report property 138
- ToString method 135
- TBaseLogBuilder Class 132

8.5.1.4 ToString

Returns textual representation of class instance.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
public
function ToString: String;
   override;
```

Return value

Value of TBaseLogBuilder.Report 138 property.

Remarks

Returns bug report. Calling this method is the same as accessing TBaseLogBuilder.Report 138 property.

See also

- Report property 138
- BuildReport method 133
- TBaseLogBuilder Class 132

8.5.2 Properties

FBuilded 136	Specifies if bug report was created or not.
CallStack 136	Call stack.
CustomFields 137	Custom fields for `Options.soCustomData`.
ExceptionInfo 137	Exception information.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Options 138	EurekaLog options.
Report 138	Generates and returns bug report.

See also

- TBaseLogBuilder 132
- ELogBuilder 90

8.5.2.1 FBuilded

Specifies if bug report was created or not.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
protected
FBuilded: Boolean;
```

Value

`True`: bug report was created, TBaseLogBuilder.Report 138 property will return cached created bug report from internal field. `False`: bug report was not created, TBaseLogBuilder.Report 138 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 138 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 134 method.

Note

Caching is performed automatically. You don't need to do anything special in TBaseLogBuilder.BuildReport 133 method. TBaseLogBuilder class will automatically cache bug report for later use.

See also

- TBaseLogBuilder Class 132

8.5.2.2 CallStack

Call stack.

Unit

ELogBuilder 90

Value

Call stack of exception. Default is TBaseLogBuilder.ExceptionInfo 137.CallStack 165.

Remarks

This property can be used by TBaseLogBuilder.BuildReport 133 method. This property can be overriden by

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

sub-classes to customize bug report content without affecting exception information.

See also

- ExceptionInfo property 137
- TEurekaExceptionInfo class 157
- BuildReport method 133
- TBaseLogBuilder Class 132

8.5.2.3 CustomFields

Custom fields for `Options.soCustomData`.

Unit

ELogBuilder 90

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 138 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 134 method to reflect changes.

See also

- BuildReport method 133
- Refresh method 134
- TBaseLogBuilder Class 132

8.5.2.4 ExceptionInfo

Exception information.

Unit

ELogBuilder 90

Value

Exception information to build bug report.

Remarks

This is exception information object which was passed to constructor 133. TBaseLogBuilder.BuildReport 133 method can use this property to gather information about exception.

Note
There are TBaseLogBuilder.CallStack 136 and TBaseLogBuilder.Options 138 properties which should be used instead of TBaseLogBuilder.ExceptionInfo.CallStack 165 and TBaseLogBuilder.ExceptionInfo.Options 174 property. TBaseLogBuilder.CallStack 136 and TBaseLogBuilder.Options 138 properties can be overriden by sub-classes to customize bug report content without affecting exception information.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEurekaExceptionInfo class 157
- Options property 138
- CallStack property 136
- Report property 138
- BuildReport method 133
- TBaseLogBuilder Class 132

8.5.2.5 Options

EurekaLog options.

Unit

ELogBuilder 90

Value

Call stack of exception. Default is TBaseLogBuilder.ExceptionInfo 137.Options 174.

Remarks

This property can be used by TBaseLogBuilder.BuildReport 133 method. This property can be overriden by sub-classes to customize bug report content without affecting exception information.

See also

- ExceptionInfo property 137
- TEurekaExceptionInfo class 157
- BuildReport method 133
- TBaseLogBuilder Class 132

8.5.2.6 Report

Generates and returns bug report.

Unit

ELogBuilder 90

Value

Textual representation of bug report.

Remarks

Call this property to generate and retrieve bug report for exception information which was specified in constructor 133 of log builder instance. Specified exception information can be checked by reading TBaseLogBuilder.ExceptionInfo 137 property.

Note
Actual bug report generation takes place in TBaseLogBuilder.BuildReport 133 method.

See also

- ToString method 135
- BuildReport method 133
- Refresh method 134
- TBaseLogBuilder Class 132

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.6 TConsoleWriterDialog

Enter topic text here.

8.7 TDummyLogBuilder

Dummy log builder class.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
TDummyLogBuilder = class(TBaseLogBuilder 132);
```

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 586) - because there is no way to pass "report string" to alternative constructor), but it is usefull to use already generated bug report in dialogs.

8.8 TELAbstractMapParser

Base class for .map file parsers.

Unit

EMapParserBase 95

Syntax

Code (Delphi)

```
TELAbstractMapParser = class(TObject);
```

Remarks

Methods 139	Methods
protected	Protected members
public	Public members

8.8.1 Methods

DecodeIdent 140	Returns humanreadable string from ident, returned by one of

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	MapStringToXYZ functions.
MapStringToFileName 140	Reads file name from mapped image of .map file.
MapStringToModuleName 141	Reads unit name from mapped image of .map file.
MapStringToStr 142	Reads string from mapped image of .map-file.
PrepareStr 143	Converts string from .map file to Delphi string.

See also

- TELAbstractMapParser 139
- EMapParserBase 95

8.8.1.1 DecodeIdent

Returns humanreadable string from ident, returned by one of MapStringToXYZ functions.

Unit

EMapParserBase 95

Syntax

Code (Delphi)

```
public
class function DecodeIdent(
   const AIdent: String
): String;
```

Parameters

AIdent

Result of any MapStringToXYZ method.

See also

- TELAbstractMapParser Class 139

8.8.1.2 MapStringToFileName

Reads file name from mapped image of .map file.

Unit

EMapParserBase 95

Syntax

Code (Delphi)

© 2001-2015, 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\Ìîè äîêóìåíò
// S = 'C:\Documents and Settings\user\Ìîè äîêóìåíòû\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 139

8.8.1.3 MapStringToModuleName

Reads unit name from mapped image of .map file.

Unit

EMapParserBase 95

Syntax

Code (Delphi)

```
public
class function MapStringToModuleName(
   AMapString: PELMapString
): String; virtual;
```

Parameters

AMapString

A pointer to the beginning of the string to read.

© 2001-2015, 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 139

8.8.1.4 MapStringToStr

Reads string from mapped image of .map-file.

Unit

EMapParserBase 95

Syntax

Code (Delphi)

```
protected
class function MapStringToStr(
   AMapString: PELMapString;
   const StopOnSpace: 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.

© 2001-2015, 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 139

8.8.1.5 PrepareStr

Converts string from .map file to Delphi string.

Unit

EMapParserBase 95

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.

© 2001-2015, 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 139

8.9 TELDebugInfoExports

Enter topic text here.

8.10 TELDebugInfoJedi

Enter topic text here.

8.11 TELDebugInfoMadExcept

Enter topic text here.

8.12 TELDebugInfoMap

Enter topic text here.

8.13 TELDebugInfoNative

Enter topic text here.

8.14 TELDebugInfoSymbols

Enter topic text here.

8.15 TELDebugInfoTD32

Enter topic text here.

8.16 TELFTPSender

Implements sending via FTP uploading.

Unit

ESendWebFTP 111

Syntax

Code (Delphi)

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

Remarks

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.17 TELHTTPSender

Implements sending via HTTP posting.

Unit
ESendWebHTTP 111

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 108

Syntax

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

Remarks
Implemented by Outlook.

8.19 TELMailShellSender

Implements sending via mailto: protocol

Unit
ESendMailShell 109

Syntax

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

Remarks
Works via ShellExecute.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.20 TELMailSimpleMAPISender

Implements sending via Simple MAPI

Unit
ESendMailSMAPI 109

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 110

Syntax

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

Remarks

8.22 TELMailSMTPClientSender

Implements sending via SMTP client protocol.

Unit
ESendMailSMTP 110

Syntax

Code (Delphi)
```
TELMailSMTPClientSender = class(TELMailSMTPBaseSender 146);
```

Remarks
Same as normal e-mail client application.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.23 TELMailSMTPSender

Unit
ESendMailSMTP 110

Syntax

Code (Delphi)
```
TELMailSMTPSender = class(TELMailSMTPBaseSender 146);
```

Remarks

8.24 TELMailSMTPServerSender

Implements sending via SMTP server protocol.

Unit
ESendMailSMTP 110

Syntax

Code (Delphi)
```
TELMailSMTPServerSender = class(TELMailSMTPBaseSender 146);
```

Remarks
Works by creating a fake e-mail server.

8.25 TELTrackerBugZillaSender

Enter topic text here.

8.26 TELTrackerBugZillaSender

Implements API access to BugZilla bug tracker

Unit
ESendAPIBugZilla 106

Syntax

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

Remarks
BugZilla uses XML-RPC API.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Requirements

Bug tracker	BugZilla 3 or above

8.27 TELTrackerFogBugzSender

Implements API access to Fogbugz bug tracker

Unit

ESendAPIFogBugz 106

Syntax

Code (Delphi)

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

Remarks

Fogbugz uses custom XML API.

Requirements

Bug tracker	Fogbugz 6 or above

8.28 TELTrackerFogBugzSender

Enter topic text here.

8.29 TELTrackerJIRASender

Implements API access to JIRA bug tracker

Unit

ESendAPIJIRA 107

Syntax

Code (Delphi)

```
TELTrackerJIRASender = class(TELTrackerSender);
```

Remarks

JIRA uses REST JSON API.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Requirements

Bug tracker	JIRA 5 or above

8.30 TELTrackerMantisSender

Implements API access to Mantis bug tracker

Unit
ESendAPIMantis 107

Syntax

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

Remarks
Mantis uses SOAP API.

Requirements

Bug tracker	Mantis 1.2.7 or above

8.31 TELTrackerMantisSender

Enter topic text here.

8.32 TEurekaBaseStackList

Base call stack class.

Unit
ECallStack 54

Syntax

Code (Delphi)
```
TEurekaBaseStackList = class(TInterfacedPersistent);
```

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 584 to create default call stack class or use GetTracerClass 332 to get specific class.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Methods 150	Methods
public	Public members

See also

- TEurekaStackListV6 class 188
- TEurekaStackListV7 class 189
- TGenericStackList class 220
- EurekaCallStackClass global variable 584
- GetTracer function 331
- GetTracerClass function 332

8.32.1 Methods

GetCurrentInstruction 150	Returns current instraction for current thread.
GetCurrentStackFrame 151	Returns current CPU stack frame for current thread.
GetCurrentStackPointer 152	Returns current CPU stack pointer for current thread.
GetCurrentTopOfStack 152	Returns top of CPU stack for current thread.
Create 153	Creates call stack object instance and (optionally) builds/traces stack.
Build 155	Builds call stack.

See also

- TEurekaBaseStackList 149
- ECallStack 54

8.32.1.1 GetCurrentInstruction

Returns current instraction for current thread.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
public
class function GetCurrentInstruction: Pointer;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

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 152
- GetCurrentStackPointer method 152
- GetCurrentStackFrame method 151
- TEurekaBaseStackList Class 149

8.32.1.2 GetCurrentStackFrame

Returns current CPU stack frame for current thread.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
public
class function GetCurrentStackFrame: ETypes.PStackFrame;
```

Return value

Pointer to stack's data, which points to current stack frame.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This value is very similar to GetCurrentStackPointer 152, 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 331.

See also

- GetCurrentTopOfStack method 152
- GetCurrentStackFrame method 151
- GetCurrentInstruction method 150
- TEurekaBaseStackList Class 149

8.32.1.3 GetCurrentStackPointer

Returns current CPU stack pointer for current thread.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
public
class function GetCurrentStackPointer: Pointer;
```

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 152
- GetCurrentStackFrame method 151
- GetCurrentInstruction method 150
- TEurekaBaseStackList Class 149

8.32.1.4 GetCurrentTopOfStack

Returns top of CPU stack for current thread.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
public
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
class function GetCurrentTopOfStack: Pointer;
```

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 151
- GetCurrentInstruction method 150
- TEurekaBaseStackList Class 149

8.32.1.5 Create

Creates call stack object instance and (optionally) builds/traces stack.

Unit

ECallStack 54

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 = [ddUnit, ddProcedure, ddSc
   const AThreadID: DWord = 0;
   const ARunningThread: Boolean = True;
   const AGetDebugInfo: Boolean = True;
   const AShowBeginCalls: Boolean = False;
   const AAllowedMethods: DWord = TracerInvalidMask;
   const AStackOffset: PtrUInt = 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.
- 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 582 (which is "EurekaLog 7 RAW"). Do not use this argument. Use GetTracer 331 or GetTracerClass 332 functions.

Remarks

This is object constructor, which creates instances of call stack.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Warning

Do not create instances of TEurekaBaseStackList class. Use EurekaCallStackClass 584 to create default call stack class or use GetTracerClass 332 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 155 method. `Create` with arguments will create object instance AND build call stack. This overload is equal to `Create` + `Build`.

See also

- TEurekaBaseStackList Class 149

8.32.1.6 Build

Builds call stack.

Unit

ECallStack 54

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 = [ddProcedure, ddUnit, ddSc
   const AThreadID: DWord = 0;
   const ARunningThread: Boolean = True;
   const AGetDebugInfo: Boolean = True;
   const AShowBeginCalls: Boolean = False;
   const AMethods: DWord = TracerInvalidMask;
   const AStackOffset: PtrUInt = 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).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.
- 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`.

`AMethods` [in, optional]
Mask for call stack tracing methods to use. Default is DefaultTracerMask 582 (which is "EurekaLog 7 RAW"). Do not use this argument. Use GetTracer 331 or GetTracerClass 332 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).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

You can walk though stack by using Count and Items properties.

See also

- BuildForThread method
- BuildForThreadByID Method
- TEurekaBaseStackList Class 149

8.33 TEurekaExceptionInfo

Exception information class.

Unit

EException 81

Syntax

Code (Delphi)

```
TEurekaExceptionInfo = class(TInterfacedPersistent);
```

Remarks

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

Methods 157	Methods
Properties 158	Properties
public	Public members

See also

- ExceptionManager function 305

8.33.1 Methods

ToString 157	String representation of exception information.

See also

- TEurekaExceptionInfo 157
- EException 81

8.33.1.1 ToString

String representation of exception information.

Unit

EException 81

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
public
function ToString: String;
   override;
```

Return value

Exception message and compact call stack.

Remarks

Use this property to get simple human-redable description of the exception.

Tip

Use ELogBuilder 90 and/or ESysInfo 115 units to build more advanced logs.

See also

- ExceptionManager 305
- TEurekaExceptionInfo Class 157

8.33.2 Properties

ActiveControl 160	Window handle of active control.
ActiveControlClass 160	Class name of active control.
ActiveControlText 161	Text of active control.
ActiveForm 161	Window handle of active window.
ActiveFormClass 162	Class name of active window.
ActiveFormText 162	Text of active window.
AdditionalInfo 162	Custom data.
Address 163	Exception address.
BaseException 163	Base exception.
BugID 164	BugID value.
BugIDSource 164	Source information for BugID.
BugIDStr 165	String representation of BugID property.
CallStack 165	Exception call stack.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Context 165	CPU state.
Cursor 166	Mouse cursor.
CursorPos 166	Mouse cursor position.
CustomID 167	Custom part of BugID value.
ExceptionClass 167	Exception class.
ExceptionMessage 167	Exception message.
ExceptionNumber 168	Number of exceptions.
ExceptionObject 168	Exception object.
ExceptionRecord 169	Exception record.
ExpectedContext 169	Help context number for exception.
ExpectedURL 170	Exception URL.
Handled 170	Indicates type of exception (handled).
Handler 171	Exception handler name.
Info 171	Returns low-level exception information.
InitializationException 172	Indicates type of exception (initialization).
InnerException 172	Inner exception.
Module 173	Exception module.
ModuleName 173	Exception module name.
ModuleVersion 173	Version of exception module.
Options 174	EurekaLog options for this exception.
Processed 174	Indicates type of exception (processed).
Reader 175	Exception information reader.
ReproduceText 175	Steps to reproduce.
SafeCallExpt 176	Indicates type of exception (safecall).
StackDump 176	Stack dump.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

StackPoint 177	Indicates current position of stack.
StackTop 177	Indicates top of stack.
SynchronizeException 178	Indicates type of exception (synchronize).
ThreadException 178	Indicates type of exception (thread).
ThreadID 179	No description available
Time 179	Exception time.
UnhandledExceptionNumber 179	Number of unhandled exceptions.

See also

- TEurekaExceptionInfo 157
- EException 81

8.33.2.1 ActiveControl

Window handle of active control.

Unit

EException 81

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 162
- ActiveFormText property 162
- ActiveControlClass property 160
- ActiveControlText property 161
- ActiveForm property 161
- TEurekaExceptionInfo Class 157

8.33.2.2 ActiveControlClass

Class name of active control.

Unit

EException 81

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Value
Window class name. Such as "EDIT".

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormClass property 162
- ActiveFormText property 162
- ActiveControlText property 161
- ActiveControl property 160
- TEurekaExceptionInfo Class 157

8.33.2.3 ActiveControlText

Text of active control.

Unit
EException 81

Value
Arbitrary text from active control.

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormClass property 162
- ActiveFormText property 162
- ActiveControlClass property 160
- ActiveControl property 160
- TEurekaExceptionInfo Class 157

8.33.2.4 ActiveForm

Window handle of active window.

Unit
EException 81

Value
Window handle of 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. Window may be deleted right after exception was raised.

See also
- ActiveFormClass property 162
- ActiveFormText property 162

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ActiveControlClass property 160
- ActiveControlText property 161
- ActiveControl property 160
- TEurekaExceptionInfo Class 157

8.33.2.5 ActiveFormClass

Class name of active window.

Unit
EException 81

Value
Window class name. Such as "TForm1".

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormText property 162
- ActiveControlClass property 160
- ActiveControlText property 161
- ActiveForm property 161
- TEurekaExceptionInfo Class 157

8.33.2.6 ActiveFormText

Text of active window.

Unit
EException 81

Value
Arbitrary text from active window.

Remarks
Contains information about GUI elements when exception was raised.

See also
- ActiveFormClass property 162
- ActiveControlClass property 160
- ActiveControlText property 161
- ActiveForm property 161
- TEurekaExceptionInfo Class 157

8.33.2.7 AdditionalInfo

Custom data.

Unit
EException 81

Value
Arbitrary user-supplied data for the exception.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 167 for dialogs. See UseExceptionComments option.

See also

- TEurekaExceptionInfo Class 157

8.33.2.8 Address

Exception address.

Unit

EException 81

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.

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

- CallStack property 165
- Module property 173
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.9 BaseException

Base exception.

Unit

EException 81

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

exceptions.

See also

- InnerException property 172
- TEurekaExceptionInfo Class 157

8.33.2.10 BugID

BugID value.

Unit

EException 81

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 164") 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 167 to alter user part of BugID value.

Tip

Use BugIDStr 165 to get human-readable string representation of BugID value.

See also

- CustomID property 167
- BugIDStr property 165
- BugIDSource property 164
- TEurekaExceptionInfo Class 157

8.33.2.11 BugIDSource

Source information for BugID.

Unit

EException 81

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 164
- TEurekaExceptionInfo Class 157

8.33.2.12 BugIDStr

String representation of BugID property.

Unit

EException 81

Value

Hex-encoded BugID property.

Remarks

Use this property to get human-readable string representation of BugID 164

See also

- BugID property 164
- TEurekaExceptionInfo Class 157

8.33.2.13 CallStack

Exception call stack.

Unit

EException 81

Value

Call stack for exception.

Remarks

This call stack includes address of exception as first item.

See also

- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.14 Context

CPU state.

Unit

EException 81

Value

Contains CPU information for exception.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 169
- Reader property 175
- Address property 163
- CallStack property 165
- ExceptionObject property 168
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.15 Cursor

Mouse cursor.

Unit

EException 81

Value

Cursor descriptor.

Remarks

Indicates mouse cursor shape when exception was raised.

See also

- CursorPos property 166
- TEurekaExceptionInfo Class 157

8.33.2.16 CursorPos

Mouse cursor position.

Unit

EException 81

Value

Absolute desktop coordinates of mouse cursor position.

Remarks

Indicates mouse position when exception was raised.

See also

- Cursor property 166
- TEurekaExceptionInfo Class 157

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.33.2.17 CustomID

Custom part of BugID value.

Unit
EException 81

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 164
- TEurekaExceptionInfo Class 157

8.33.2.18 ExceptionClass

Exception class.

Unit
EException 81

Value
Class of exception object (if present). Examples: 'Exception', 'EAccessViolation', 'EStreamError', etc.

Remarks
Class name of original exception object.

See also
- ExceptionMessage property 167
- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.19 ExceptionMessage

Exception message.

Unit
EException 81

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 162 property.

See also

- AdditionalInfo property 162
- ExceptionClass property 167
- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.20 ExceptionNumber

Number of exceptions.

Unit

EException 81

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 179
- TExceptionManager.ExceptionCount 215
- TEurekaExceptionInfo Class 157

8.33.2.21 ExceptionObject

Exception object.

Unit

EException 81

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, 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.

Use Address 163, ExceptionClass 167 and ExceptionMessage 167 properties instead.

See also

- TEurekaExceptionInfo Class 157

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.33.2.22 ExceptionRecord

Exception record.

Unit

EException 81

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 165
- Reader property 175
- Address property 163
- CallStack property 165
- ExceptionObject property 168
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.23 ExpectedContext

Help context number for exception.

Unit

EException 81

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 389 routine.

Tip
This property is used by "Help" button in dialogs. Setting this property will force "Help" button to appear.

© 2001-2015, 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 170
- TEurekaExceptionInfo Class 157

8.33.2.24 ExpectedURL

Exception URL.

Unit

EException 81

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 389 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 170
- TEurekaExceptionInfo Class 157

8.33.2.25 Handled

Indicates type of exception (handled).

Unit

EException 81

Value

`True` - this is handled exception.

`False` - this is unhandled exception.

Remarks

This property indicates type of exception - handled or not. Handled = `False` on creation and may become

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

`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 174
- SafeCallExpt property
- SynchronizeException property 178
- InitializationException property 172
- ThreadException property 178
- TEurekaExceptionInfo Class 157

8.33.2.26 Handler

Exception handler name.

Unit

EException 81

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 157

8.33.2.27 Info

Returns low-level exception information.

Unit

EException 81

Value

Low-level exception information.

Remarks

Low-level information that was captured for this exception.

Tip
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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ExceptionRecord property 169
- Context property 165
- Reader property 175
- TEurekaExceptionInfo Class 157

8.33.2.28 InitializationException

Indicates type of exception (initialization).

Unit

EException 81

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 174
- Handled property 170
- SafeCallExpt property 176
- SynchronizeException property 178
- ThreadException property 178
- TEurekaExceptionInfo Class 157

8.33.2.29 InnerException

Inner exception.

Unit

EException 81

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
Unlike RTL's InnerException, EurekaLog's `InnerException` is always captured.

See also

- BaseException property 163
- TEurekaExceptionInfo Class 157

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.33.2.30 Module

Exception module.

Unit
EException 81

Value
HINST/HMODULE of executable module containing exception address 163. 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 173 / ModuleVersion 173 properties to get module information.

See also
- ModuleName property 173
- ModuleVersion property 173
- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.31 ModuleName

Exception module name.

Unit
EException 81

Value
Full file name of executable module containing exception address 163. 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 173
- ModuleVersion property 173
- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.32 ModuleVersion

Version of exception module.

Unit
EException 81

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Value

Version information of executable module containing exception address 163. 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 173
- ModuleName property 173
- Address property 163
- TEurekaExceptionInfo Class 157

8.33.2.33 Options

EurekaLog options for this exception.

Unit

EException 81

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 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 157

8.33.2.34 Processed

Indicates type of exception (processed).

Unit

EException 81

Value

`True` - this is processed exception.

`False` - this is unprocessed exception.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 305. Handle 205 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 170
- SafeCallExpt property 176
- SynchronizeException property 178
- InitializationException property 172
- ThreadException property 178
- TEurekaExceptionInfo Class 157

8.33.2.35 Reader

Exception information reader.

Unit

EException 81

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 `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 169
- Context property 165
- Address property 163
- CallStack property 165
- ExceptionObject property 168
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.36 ReproduceText

Steps to reproduce.

Unit

EException 81

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 157

8.33.2.37 SafeCallExpt

Indicates type of exception (safecall).

Unit

EException 81

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).

See also

- Processed property 174
- Handled property 170
- SynchronizeException property 178
- InitializationException property 172
- ThreadException property 178
- TEurekaExceptionInfo Class 157

8.33.2.38 StackDump

Stack dump.

Unit

EException 81

Value

Pointer to stack dump.

Remarks

Stack dump is a block of RAW memory. It's copied from CPU stack from StackPoint 177 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.

See also

- StackPoint property 177
- CallStack property 165
- Address property 163

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- TEurekaExceptionInfo Class 157

8.33.2.39 StackPoint

Indicates current position of stack.

Unit
EException 81

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 177
- ExceptionRecord property 169
- Context property 165
- Address property 163
- CallStack property 165
- ExceptionObject property 168
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.40 StackTop

Indicates top of stack.

Unit
EException 81

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.

See also
- StackPoint property 177
- ExceptionRecord property 169
- Context property 165
- Address property 163

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- CallStack property 165
- ExceptionObject property 168
- ExceptionClass property 167
- ExceptionMessage property 167
- TEurekaExceptionInfo Class 157

8.33.2.41 SynchronizeException

Indicates type of exception (synchronize).

Unit

EException 81

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

- Processed property 174
- Handled property 170
- SafeCallExpt property 176
- InitializationException property 172
- ThreadException property 178
- TEurekaExceptionInfo Class 157

8.33.2.42 ThreadException

Indicates type of exception (thread).

Unit

EException 81

Value

`True` - this exception was raised inside background thread and it is captured to be re-raised later (in main thread).

`False` - this is normal exception.

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Processed property 174
- Handled property 170
- SafeCallExpt property 176
- InitializationException property 172
- SynchronizeException property 178
- TEurekaExceptionInfo Class 157

8.33.2.43 ThreadID

Thread ID (actual exception's thread).

Unit

EException 81

Value

ID of thread. It identificates thread that was 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.

See also

- Address property 163
- CallStack property 165
- TEurekaExceptionInfo Class 157

8.33.2.44 Time

Exception time.

Unit

EException 81

Value

Date-time of exception raising.

Remarks

Indicates when exception was raised.

See also

- TEurekaExceptionInfo Class 157

8.33.2.45 UnhandledExceptionNumber

Number of unhandled exceptions.

Unit

EException 81

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 168
- TExceptionManager.UnhandledExceptionCount 216
- TEurekaExceptionInfo Class 157

8.34 TEurekaLogDetailedDialog

Enter topic text here.

8.35 TEurekaLogDialog

Enter topic text here.

8.36 TEurekaLogEvents

EurekaLog 7 component.

Unit

EComponent 56

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 75 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 75 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Events 30	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

8.36.1 Events

OnAttachedFilesRequest 182	Wrapper for OnAttachedFilesRequest 19 event.
OnBeginReportGen 182	Wrapper for OnBeginReportGen 26 event.
OnCustomBugID 183	Wrapper for OnCustomBugID 22 event.
OnCustomButtonClick 183	Wrapper for OnCustomButtonClick 24 event.
OnCustomDataRequest 183	Wrapper for OnCustomDataRequest 23 event.
OnCustomFileName 184	Wrapper for OnCustomFileName 21 event.
OnCustomWebFieldsRequest 184	Wrapper for OnCustomWebFieldsRequest 23 event.
OnEndReportGen 184	Wrapper for OnEndReportGen 26 event.
OnExceptionAction 185	Wrapper for OnExceptionAction 15 event.
OnExceptionError 185	Wrapper for OnExceptionError 17 event.
OnExceptionNotify 186	Wrapper for OnExceptionNotify 14 event.
OnPasswordRequest 186	Wrapper for OnPasswordRequest 18 event.
OnRaise 186	Wrapper for OnRaise 12 event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

OnReproduceRequest 187	Wrapper for OnReproduceRequest 21 event.
OnZippedFilesRequest 187	Wrapper for OnZippedFilesRequest 20 event.

See also
- TEurekaLogEvents 29
- EComponent 56

8.36.1.1 OnAttachedFilesRequest

Wrapper for OnAttachedFilesRequest 19 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnAttachedFilesRequest event 19.

Examples
For code example please see OnAttachedFilesRequest event 19.

See also
- Events 11
- OnAttachedFilesRequest event 19
- TEurekaLogEvents Class 29

8.36.1.2 OnBeginReportGen

Wrapper for OnBeginReportGen 26 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnBeginReportGen event 26.

Examples
For code example please see OnBeginReportGen event 26.

See also
- Events 11
- OnBeginReportGen event 26
- TEurekaLogEvents Class 29

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.36.1.3 OnCustomBugID

Wrapper for OnCustomBugID 22 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnCustomBugID event 22.

Examples
For code example please see OnCustomBugID event 22.

See also
- Events 11
- OnCustomBugID event 22
- TEurekaLogEvents Class 29

8.36.1.4 OnCustomButtonClick

Wrapper for OnCustomButtonClick 24 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnCustomButtonClick event 24.

Examples
For code example please see OnCustomButtonClick event 24.

See also
- Events 11
- OnCustomButtonClick event 24
- TEurekaLogEvents Class 29

8.36.1.5 OnCustomDataRequest

Wrapper for OnCustomDataRequest 23 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnCustomDataRequest event 23.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Examples

For code example please see OnCustomDataRequest event 23.

See also

- Events 11
- OnCustomDataRequest event 23
- TEurekaLogEvents Class 29

8.36.1.6 OnCustomFileName

Wrapper for OnCustomFileName 21 event.

Unit

EComponent 56

Value

User-supplied event handler.

Remarks

For the detailed description please see OnCustomFileName event 21.

Examples

For code example please see OnCustomFileName event 21.

See also

- Events 11
- OnCustomFileName event 21
- TEurekaLogEvents Class 29

8.36.1.7 OnCustomWebFieldsRequest

Wrapper for OnCustomWebFieldsRequest 23 event.

Unit

EComponent 56

Value

User-supplied event handler.

Remarks

For the detailed description please see OnCustomWebFieldsRequest event 23.

Examples

For code example please see OnCustomWebFieldsRequest event 23.

See also

- Events 11
- OnCustomWebFieldsRequest event 23
- TEurekaLogEvents Class 29

8.36.1.8 OnEndReportGen

Wrapper for OnEndReportGen 26 event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnEndReportGen event 26.

Examples
For code example please see OnEndReportGen event 26.

See also
- Events 11
- OnEndReportGen event 26
- TEurekaLogEvents Class 29

8.36.1.9 OnExceptionAction

Wrapper for OnExceptionAction 15 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnExceptionAction event 15.

Examples
For code example please see OnExceptionAction event 15.

See also
- Events 11
- OnExceptionAction event 15
- TEurekaLogEvents Class 29

8.36.1.10 OnExceptionError

Wrapper for OnExceptionError 17 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnExceptionError event 17.

Examples
For code example please see OnExceptionError event 17.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Events 11
- OnExceptionError event 17
- TEurekaLogEvents Class 29

8.36.1.11 OnExceptionNotify

Wrapper for OnExceptionNotify 14 event.

Unit

EComponent 56

Value

User-supplied event handler.

Remarks

For the detailed description please see OnExceptionNotify event 14.

Examples

For code example please see OnExceptionNotify event 14.

See also

- Events 11
- OnExceptionNotify event 14
- TEurekaLogEvents Class 29

8.36.1.12 OnPasswordRequest

Wrapper for OnPasswordRequest 18 event.

Unit

EComponent 56

Value

User-supplied event handler.

Remarks

For the detailed description please see OnPasswordRequest event 18.

Examples

For code example please see OnPasswordRequest event 18.

See also

- Events 11
- OnPasswordRequest event 18
- TEurekaLogEvents Class 29

8.36.1.13 OnRaise

Wrapper for OnRaise 12 event.

Unit

EComponent 56

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 29

8.36.1.14 OnReproduceRequest

Wrapper for OnReproduceRequest 21 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnReproduceRequest event 21.

Examples
For code example please see OnReproduceRequest event 21.

See also
- Events 11
- OnReproduceRequest event 21
- TEurekaLogEvents Class 29

8.36.1.15 OnZippedFilesRequest

Wrapper for OnZippedFilesRequest 20 event.

Unit
EComponent 56

Value
User-supplied event handler.

Remarks
For the detailed description please see OnZippedFilesRequest event 20.

Examples
For code example please see OnZippedFilesRequest event 20.

See also
- Events 11

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- OnZippedFilesRequest event 20
- TEurekaLogEvents Class 29

8.37 TEurekaModuleOptions

EurekaLog options.

Unit
EClasses 55

Syntax

Code (Delphi)

```
TEurekaModuleOptions = class(TInterfacedPersistent);
```

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.

8.38 TEurekaStackListV6

Stack tracing class for EurekaLog 6 method.

Unit
ECallStack 54

Syntax

Code (Delphi)

```
TEurekaStackListV6 = class(TEurekaBaseStackList 149);
```

Remarks
Use this class to build call stacks by using EurekaLog 6 method.

Tip

Avoid using classes directly. Use EurekaCallStackClass 584 / GetTracer 331 / GetTracerClass 332.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
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 189
- TGenericStackList class 220

8.39 TEurekaStackListV7

Stack tracing class for EurekaLog 7 RAW method.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
TEurekaStackListV7 = class(TEurekaBaseStackList 149);
```

Remarks

Use this class to build call stacks by using EurekaLog 7 RAW method.

Tip

Avoid using classes directly. Use EurekaCallStackClass 584 / GetTracer 331 / GetTracerClass 332.

Examples

Code (Delphi)

```
var
   CallStack: TEurekaBaseStackList;
begin
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   CallStack := GetTracer(TracerRawEurekaLogV7); // preferable
    // Or:
    // CallStack := TEurekaStackListV7.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

- TEurekaStackListV6 class 188
- TGenericStackList class 220

8.40 TEurekaThreadData

Thread information.

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
TEurekaThreadData = class(TObject);
```

Remarks

This class represents information about running thread in your application. Instances of this class can be acquired via ThreadData 452, AcquireThreadData or ThreadDataEnum 453 functions.

Methods 190	Methods
Properties 191	Properties
public	Public members

See also

- ThreadData Function 452
- AcquireThreadData Function
- ThreadDataEnum Function 453
- TEurekaThreadDataList Class 200

8.40.1 Methods

Priority 191	No description available

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- TEurekaThreadData 190
- EThreadsManager 117

8.40.1.1 Priority

Thread's priority.

Unit

EThreadsManager 117

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 190

8.40.2 Properties

CallerAddress 192	Thread creator (address).
CallerFunction 193	Thread creator (description).
Handle 193	Thread handle.
ID 194	TID.
IsMainThread 194	Indicate whenever this thread is main thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

IsRTLThread 194	Indicate whenever this thread is RTL thread.
IsWindowsThread 195	Indicate whenever this thread is Windows thread.
Kind 195	Thread kind (Windows/RTL thread).
Name 196	Thread name.
Parameter 196	No description available
ParentID 197	Parent thread ID.
RealParameter 197	No description available
RealStartingAddress 197	Real thread function (pointer).
RealStartingFunction 198	Real thread function (description).
StackTop 198	No description available
StartingAddress 199	Usual thread function (pointer).
StartingFunction 199	Usual thread function (description).
Thread 200	Thread object (class instance).
ThreadClassName 200	Thread class name.

See also

- TEurekaThreadData 190
- EThreadsManager 117

8.40.2.1 CallerAddress

Thread creator (address).

Unit

EThreadsManager 117

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- RealStartingAddress Property 197
- CallerFunction Property 193
- Thread Property 200
- ThreadClassName Property 200
- TEurekaThreadData Class 190

8.40.2.2 CallerFunction

Thread creator (description).

Unit

EThreadsManager 117

Value

A human-readable description of CallerAddress 192 property.

Remarks

This value contains function name identified in CallerAddress 192 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 198
- CallerAddress Property 192
- Thread Property 200
- ThreadClassName Property 200
- TEurekaThreadData Class 190

8.40.2.3 Handle

Thread handle.

Unit

EThreadsManager 117

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 194 property to identify threads.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ID Property 194
- ParentID Property 197
- Name Property 196
- TEurekaThreadData Class 190

8.40.2.4 ID

TID.

Unit

EThreadsManager 117

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 197
- Handle Property 193
- Name Property 196
- IsMainThread Property 194
- TEurekaThreadData Class 190

8.40.2.5 IsMainThread

Indicate whenever this thread is main thread.

Unit

EThreadsManager 117

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 195
- IsRTLThread Property 194
- IsWindowsThread Property 195
- TEurekaThreadData Class 190

8.40.2.6 IsRTLThread

Indicate whenever this thread is RTL thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EThreadsManager 117

Value
True for RTL threads, false for Windows threads.

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 195 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 195
- IsMainThread Property 194
- IsWindowsThread Property 195
- TEurekaThreadData Class 190

8.40.2.7 IsWindowsThread

Indicate whenever this thread is Windows thread.

Unit
EThreadsManager 117

Value
`True` for Windows threads, `False` for RTL threads.

Remarks
This property is the opposite for IsRTLThread 194 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also
- Kind Property 195
- IsRTLThread Property 194
- IsMainThread Property 194
- TEurekaThreadData Class 190

8.40.2.8 Kind

Thread kind (Windows/RTL thread).

Unit
EThreadsManager 117

Value
Indicates thread kind.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Kind of the thread is detected on thread's creation. It is not changed during thread's life-time.

See also

- IsMainThread Property 194
- IsRTLThread Property 194
- IsWindowsThread Property 195
- TEurekaThreadData Class 190

8.40.2.9 Name

Thread name.

Unit

EThreadsManager 117

Value

Optional thread name.

Remarks

This is an optional thread name. Threads can be named to simplify debugging. Thread name is an arbitrary string which will be used in bug reports to describe thread. Default value is 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 386 or NameThreadForDebugging 387 functions to name a thread.

See also

- NameThread Function 386
- NameThreadForDebugging Function 387
- ID Property 194
- ParentID Property 197
- Handle Property 193
- IsMainThread Property 194
- TEurekaThreadData Class 190

8.40.2.10 Parameter

Usual thread function's argument.

Unit

EThreadsManager 117

Value

RAW pointer argument for StartingAddress 199 thread function.

Remarks

This property contains argument passed to thread function identified by StartingAddress 199 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- RealParameter Property 197
- StartingAddress Property 199
- StartingFunction Property 199
- TEurekaThreadData Class 190

8.40.2.11 ParentID

Parent thread ID.

Unit

EThreadsManager 117

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 194
- Handle Property 193
- Name Property 196
- IsMainThread Property 194
- TEurekaThreadData Class 190

8.40.2.12 RealParameter

Real thread function's argument.

Unit

EThreadsManager 117

Value

RAW pointer argument for RealStartingAddress 197 thread function.

Remarks

This property contains argument passed to thread function identified by RealStartingAddress 197 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- Parameter Property 196
- RealStartingAddress Property 197
- RealStartingFunction Property 198
- Thread Property 200
- ThreadClassName Property 200
- TEurekaThreadData Class 190

8.40.2.13 RealStartingAddress

Real thread function (pointer).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EThreadsManager 117

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 199 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- StartingAddress Property 199
- CallerAddress Property 192
- RealStartingFunction Property 198
- RealParameter Property 197
- Thread Property 200
- ThreadClassName Property 200
- ThreadClassName Property 200
- TEurekaThreadData Class 190

8.40.2.14 RealStartingFunction

Real thread function (description).

Unit

EThreadsManager 117

Value

A human-readable description of RealStartingAddress 197 property.

Remarks

This value contains function name identified in RealStartingAddress 197 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 199
- CallerFunction Property 193
- RealStartingAddress Property 197
- RealParameter Property 197
- TEurekaThreadData Class 190

8.40.2.15 StackTop

Top of thread's start.

Unit

EThreadsManager 117

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Value
A pointer to the beginning of the stack (top high address).

Remarks
This value is used for call stack creation.

See also
- TEurekaThreadData Class 190

8.40.2.16 StartingAddress

Usual thread function (pointer).

Unit
EThreadsManager 117

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 195.

A real thread function is a function passed to `CreateThread/BeginThread/TThread.Execute` functions. This value is stored in RealStartingAddress 197 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 197
- CallerAddress Property 192
- StartingFunction Property 199
- Parameter Property 196
- TEurekaThreadData Class 190

8.40.2.17 StartingFunction

Usual thread function (description).

Unit
EThreadsManager 117

Value
A human-readable description of StartingAddress 199 property.

Remarks
This value contains function name identified in StartingAddress 199 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- RealStartingFunction Property 198
- CallerFunction Property 193
- StartingAddress Property 199
- Parameter Property 196
- TEurekaThreadData Class 190

8.40.2.18 Thread

Thread object (class instance).

Unit

EThreadsManager 117

Value

A thread class instance for the thread. Typically this is a TThread instance. This value is `nil` for Windows 195 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 197
- CallerAddress Property 192
- ThreadClassName Property 200
- TEurekaThreadData Class 190

8.40.2.19 ThreadClassName

Thread class name.

Unit

EThreadsManager 117

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 200 property.

This value is detected on thread's creation. It is not changed during thread's life-time.

See also

- Thread Property 200
- TEurekaThreadData Class 190

8.41 TEurekaThreadDataList

List of threads.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EThreadsManager 117

Syntax

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

Remarks

Represents thread list returned by ThreadDataEnum 453 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 201	Properties
public	Public members

See also

- ThreadDataEnum Function 453
- TEurekaThreadData Class 190

8.41.1 Properties

Count 201	Count of threads in the list.
Items 202	Provides access to individual threads in the list.
ThreadData 202	Returns information about a particular thread.

See also

- TEurekaThreadDataList 200
- EThreadsManager 117

8.41.1.1 Count

Count of threads in the list.

Unit

EThreadsManager 117

Remarks

This property indicate count of threads in current list. Use this property to enumerate all threads via Items 202 property. If you're interested in a particular thread - use ThreadData 202 method.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Items property 202
- ThreadData Method 202
- ThreadDataEnum Function 453
- TEurekaThreadDataList Class 200

8.41.1.2 Items

Provides access to individual threads in the list.

Unit

EThreadsManager 117

Parameters

AIndex [in]

Index of a thread in the list. Index should be in 0.. Count 201-1 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 202 method.

See also

- ThreadDataEnum Function 453
- ThreadData Method 202
- Count property 201
- TEurekaThreadDataList Class 200

8.41.1.3 ThreadData

Returns information about a particular thread.

Unit

EThreadsManager 117

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 452 function to modify thread's dynamic properties.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ThreadDataEnum Function 453
- ThreadData Function 452
- TEurekaThreadData Class 190
- TEurekaThreadDataList Class 200

8.42 TEventLogDialog

Enter topic text here.

8.43 TExceptionManager

Exception manager class.

Unit

EExceptionManager 84

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 305 or TExceptionManager.Instance 204 to access `TExceptionManager` class.

Methods 203	Methods
Properties 214	Properties
protected	Protected members
public	Public members

See also

- ExceptionManager function 305
- TExceptionManager.Instance class function 204

8.43.1 Methods

Instance 204	Returns singleton instance for `TExceptionManager` class.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ClearThreadExceptions 205	Clears last occurred exception in current thread.
Handle 205	Handles (processes) specified exception.
Info 208	Returns information about exception.
Notify 209	Notifies exception manager about exception.
ShowLastExceptionData 211	Handles last active exception (if any exists).
StandardEurekaError 212	Invokes EurekaLog without raising exception.
StandardEurekaNotify 213	Handle a specific exception.

See also

- TExceptionManager 203
- EExceptionManager 84

8.43.1.1 Instance

Returns singleton instance for `TExceptionManager` class.

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
public
class function Instance: TExceptionManager 203;
```

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 203 for more information.

Note

This method is the same as ExceptionManager 305 function.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ExceptionManager function 305
- TExceptionManager Class 203

8.43.1.2 ClearThreadExceptions

Clears last occurred exception in current thread.

Unit

EExceptionManager 84

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 215 will return `nil`.
- ShowLastExceptionData 211 will do nothing (return srNoExceptionInfo 559 code).
- Corresponding TEurekaExceptionInfo 157 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 211
- LastThreadException property 215
- TEurekaExceptionInfo class 157
- TExceptionManager Class 203

8.43.1.3 Handle

Handles (processes) specified exception.

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
public
function Handle(
   const AInfo: TEurekaExceptionInfo 157;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse 547; overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
function Handle(
   const AInfo: TExceptionData;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[547]; overload;

function Handle(
   const AInfo: Pointer;
   const AAddr: Pointer = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[547]; overload;

function Handle(
   const AInfo: TObject;
   const AAddr: Pointer = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[547]; overload;

function Handle(
   const AInfo: Windows.TExceptionRecord;
   const AContext: PContext = nil;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[547]; overload;

function Handle(
   const AInfo: TExceptionPointers;
   const AHandled: Boolean = False;
   const ARespectActivation: Boolean = True
): TResponse[547]; 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[559].

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`,

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

otherwise (EurekaLog is enabled) - do the usual work.

`False` - always process the exceptions even if EurekaLog is not active.

Return value

Result of exception processing. This record hold multiple fields with information - code, messages, etc. See TResponse 547 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 559 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)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
Thread := TMyThread.Create(False);
try
   Thread.WaitFor;
   ExceptionManager.Handle(Thread.FatalException);
finally
   FreeAndNil(Thread);
end;
```

See also

- StandardEurekaNotify Method 213
- TResponse Record 547
- Info Method 208
- Notify Method 209
- LastThreadException Property 215
- ProcessException Function
- OnExceptionNotify Event 14
- TExceptionManager Class 203

8.43.1.4 Info

Returns information about exception.

Unit

EExceptionManager 84

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 157; overload;

function Info(
   const AInfo: TObject;
   const AAddr: Pointer = nil;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 157; overload;

function Info(
   const AInfo: TExceptionData;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 157; overload;

function Info(
   const AInfo: Windows.TExceptionRecord;
   const AContext: PContext = nil;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 157; overload;

function Info(
   const AInfo: TExceptionPointers;
   const AForcedNew: Boolean = False;
   const ARespectActivation: Boolean = True;
   const ADoNotCreateNew: Boolean = False
): TEurekaExceptionInfo 157; 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.

See also

- Notify method 209
- Handle method 205
- LastThreadException property 215
- TExceptionManager Class 203

8.43.1.5 Notify

Notifies exception manager about exception.

Unit

EExceptionManager 84

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
protected
procedure Notify(
   const AInfo: TEurekaExceptionInfo 157;
   const ARespectActivation: Boolean = True
); 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 (`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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

`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 215)
3. Updates last exception 215.
4. Build chained exceptions chain.
5. Removes obsolete information blocks (deleted/not active exceptions).
6. Invokes OnRaise 12 event.

See also

- Info method 208
- Handle method 205
- LastThreadException property 215
- OnRaise event 12
- TExceptionManager Class 203

8.43.1.6 ShowLastExceptionData

Handles last active exception (if any exists).

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
public
function ShowLastExceptionData: TResponse 547;
```

Return value

Result of exception processing.

Remarks

This is the same as:

Handle 205(LastThreadException 215);

If there is no active exception - srNoExceptionInfo 559 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

This method is EurekaLog-equivalent of TApplication.HandleException method.

See also

- Handle method 205
- LastThreadException property 215
- ClearThreadExceptions method 205
- StandardEurekaNotify method 213
- StandardEurekaError method 212
- TExceptionManager Class 203

8.43.1.7 StandardEurekaError

Invokes EurekaLog without raising exception.

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
public
function StandardEurekaError(
   const Error: String;
   const AOptions: TEurekaModuleOptions 188 = nil
): TResponse 547;
```

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 205 and TResponse 547 for more information.

Remarks

This function creates exception of EEurekaLogGeneralError 131 class and passes it to Handle 205 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)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
ExceptionManager.StandardEurekaError('File does not exist, unable to cor
```

See also

- StandardEurekaNotify method 213
- ShowLastExceptionData method 211
- Info method 208
- Handle method 205
- TExceptionManager Class 203

8.43.1.8 StandardEurekaNotify

Handle a specific exception.

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
public
function StandardEurekaNotify(
   const Obj: TObject;
   const Addr: Pointer = nil;
   const AOptions: TEurekaModuleOptions 188 = nil
): TResponse 547;
```

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.

Return value

Result of exception's processing. See Handle 205 and TResponse 547 for more information.

Remarks

This is a simple combination of Info 208 and Handle 205 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 211, except it allows to specify exception object.

Tip

The diffrence between this method and the Handle 205 method is this: `Handle` method will do nothing for non-raised exceptions. `StandardEurekaNotify` will create call stack for non-raised exceptions.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 212
- ShowLastExceptionData method 211
- Info method 208
- Handle method 205
- LastThreadException property 215
- ClearThreadExceptions method 205
- TExceptionManager Class 203

8.43.2 Properties

ExceptionCount 215	Count of all exceptions.
LastThreadException 215	Returns last occurred exception in current thread.
Mode 216	Indicate work mode of exception manager.
UnhandledExceptionCount 216	Count of all processed exceptions.

See also

- TExceptionManager 203
- EExceptionManager 84

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.43.2.1 ExceptionCount

Count of all exceptions.

Unit

EExceptionManager 84

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 157.ExceptionNumber 168).

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 216 property.

Tip
This value does NOT represent count of active exceptions.

See also

- UnhandledExceptionCount property 216
- TEurekaExceptionInfo.ExceptionNumber property 168
- TEurekaExceptionInfo class 157
- TExceptionManager Class 203

8.43.2.2 LastThreadException

Returns last occurred exception in current thread.

Unit

EExceptionManager 84

Value

Information about most recent raised exception for active thread.

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 208 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- ShowLastExceptionData method 211
- ClearThreadExceptions method 205
- TEurekaExceptionInfo class 157
- Handle method 205
- TExceptionManager Class 203

8.43.2.3 Mode

Indicate work mode of exception manager.

Unit

EExceptionManager 84

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 534
- TExceptionManager Class 203

8.43.2.4 UnhandledExceptionCount

Count of all processed exceptions.

Unit

EExceptionManager 84

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.

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 157 .UnhandledExceptionNumber 179).

This value holds count of processed exceptions, it's always **less or equal** to ExceptionCount 215 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 215

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- Handle method 205
- TEurekaExceptionInfo.UnhandledExceptionNumber property 179
- TEurekaExceptionInfo class 157
- TExceptionManager Class 203

8.44 TFreezeThread

Base class for freeze detection threads.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
TFreezeThread = class(TThreadEx 225);
```

Remarks

This is an abstract class to be used as base class when implementing background freeze detection thread. Override the Execute 218 method and implement your own freeze detection logic. Call RaiseFreezeException 390 once you detect freeze/hang/deadlock.

Note

FreezeThreadClass 585 variable determinates which class EurekaLog should use. Use InitCheckFreeze 337 to re-initialize freeze detection if it is already running.

Methods 217	Methods
Properties 219	Properties
protected	Protected members
public	Public members

See also

- TMainThreadFreezeDetectionThread Class 224
- TWCTFreezeDetectionThread Function 234
- TFreezeThreadClass Type 536
- FreezeThreadClass Variable 585
- InitCheckFreeze Function 337

8.44.1 Methods

ChecksPaused 218	Indicates whether freeze checks are disabled.
Execute 218	Implements actual freeze detection code.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- TFreezeThread 217
- EFreeze 86

8.44.1.1 ChecksPaused

Indicates whether freeze checks are disabled.

Unit
EFreeze 86

Syntax

Code (Delphi)
```
protected
function ChecksPaused: Boolean;
```

Return value
`True` - freeze checks are disabled; `False` - freeze checks are enabled.

Remarks
The thread's Execute 218 method and any methods that `Execute` calls should check `ChecksPaused` periodically and do nothing when it's `True`. The PauseFreezeCheck 388 procedure sets the `ChecksPaused` property to `True`.

See also
- PauseFreezeCheck Function 388
- Terminated Property
- TFreezeThread Class 217

8.44.1.2 Execute

Implements actual freeze detection code.

Unit
EFreeze 86

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 218 property to determinate if freeze detection checks are disabled.

Important!

Do not forget to call inherited `Execute` implementation in your own overridden implementations of

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
Execute.
```

See also

- TFreezeThread Class 217

8.44.2 Properties

TargetThreadHandle 219	ThreadHandle from InitCheckFreeze 337.
TargetThreadID 219	ThreadID from InitCheckFreeze 337.
TargetWindow 219	Window handle from InitCheckFreeze 337.

See also

- TFreezeThread 217
- EFreeze 86

8.44.2.1 TargetThreadHandle

ThreadHandle from InitCheckFreeze 337.

Unit

EFreeze 86

See also

- InitCheckFreeze Function 337
- TFreezeThread Class 217

8.44.2.2 TargetThreadID

ThreadID from InitCheckFreeze 337.

Unit

EFreeze 86

See also

- InitCheckFreeze Function 337
- TFreezeThread Class 217

8.44.2.3 TargetWindow

Window handle from InitCheckFreeze 337.

Unit

EFreeze 86

See also

- InitCheckFreeze Function 337
- TFreezeThread Class 217

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.45 TGenericStackList

Common stack tracing class.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
TGenericStackList = class(TEurekaBaseStackList 149);
```

Remarks

Use this class for building call stack by any method, except EurekaLog.

Warning

Do not use this class for EurekaLog stack tracing.

Tip

Avoid using classes directly. Use EurekaCallStackClass 584/ GetTracer 331/GetTracerClass 332.

Methods 221	Methods
Properties 222	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 189

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- TEurekaStackListV6 class 188

8.45.1 Methods

Create 221	Creates call stack class with specified tracer.

See also

- TGenericStackList 220
- ECallStack 54

8.45.1.1 Create

Creates call stack class with specified tracer.

Unit

ECallStack 54

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 114 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 155 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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
      CallStack.Build(CallStack.GetCurrentInstruction);

      // ... use CallStack somehow

   finally
      FreeAndNil(CallStack);
   end;
end;
```

See also

- TGenericStackList Class 220

8.45.2 Properties

Tracer 222	Indicates stack tracer used to build call stack.

See also

- TGenericStackList 220
- ECallStack 54

8.45.2.1 Tracer

Indicates stack tracer used to build call stack.

Unit

ECallStack 54

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);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   end;
end;
```

See also

- TGenericStackList Class 220

8.46 TLogBuilder

Default EurekaLog 7 log builder.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
TLogBuilder = class(TBaseLogBuilder 132);
```

Remarks

This is default log builder class for EurekaLog 7 and above. It's used as default value for LogBuilderClass 586 global variable.

There are two more build-in log builder classes: TXMLLogBuilder 235 and TDummyLogBuilder 139.

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= |
```

© 2001-2015, 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 |TApplicatic
|00000020|03 |0018FF1C|005C1742|Project4.exe|001C1742|Forms |TApplicatic
...
etc.
```

8.47 TMainThreadFreezeDetectionThread

Thread class to detects hangs in UI thread.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
TMainThreadFreezeDetectionThread = class(TFreezeThread 217);
```

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).

8.48 TMessageBoxDetailedDialog

Enter topic text here.

8.49 TMessageBoxDialog

Enter topic text here.

8.50 TMSClassicDialog

Enter topic text here.

8.51 TNullDialog

Enter topic text here.

8.52 TRTLHandlerDialog

Enter topic text here.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.53 TThreadEx

Abstract class to create separate threads of execution.

Unit

EBase 52

Syntax

Code (Delphi)

```
TThreadEx = class(TThread);
```

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 230 method by inserting the code that should execute when the thread is executed.

Important!

You **must** call the inherited implementation in your overridden Execute 230 method.

Methods 228	Methods
Properties 232	Properties
protected	Protected members
public	Public members

Examples

Code (Delphi)

```
type
   TMyThread = class(TThreadEx)
   protected
      procedure Execute; override;
   end;

procedure TMyThread.Execute;
```

© 2001-2015, 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',
```

© 2001-2015, 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 HandleExceptionInThread(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.HandleExceptionInThread;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
   // All anonymous threads are marked with FreeOnTerminate by default
end;

procedure TForm1.HandleExceptionInThread(Sender: TObject);
var
   Thread: TThread;
begin
   Thread := (Sender as TThread);
   if Assigned(Thread.FatalException) then
      HandleException(Thread.FatalException);
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;
```

© 2001-2015, 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 264
- SetEurekaLogStateInThread Function
- NameThread Function
- HandleException Function 333

8.53.1 Methods

Create 228	Creates an instance of a thread object.
Execute 230	No description available

See also

- TThreadEx 225
- EBase 52

8.53.1.1 Create

Creates an instance of a thread object.

Unit

EBase 52

Syntax

Code (Delphi)

```
public
constructor Create;
   overload;

constructor Create(
   CreateSuspended: Boolean
); overload;

constructor Create(
   const AName: String;
   const AELEnabled: Boolean = True
); overload;

constructor Create(
   CreateSuspended: Boolean;
```


© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AName: String;
   const AELEnabled: Boolean = True
); overload;
```

Parameters

CreateSuspended [in, opt]

If `CreateSuspended` is `False`, Execute 230 is called immediately after the constructor. If `CreateSuspended` is `True`, Execute 230 is not called until after the Start method is called.Default is `False`.

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. Default is `True`. This argument is ignored if you compile your application without EurekaLog.

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');

// ...

// Start thread's execution
Thread.Start;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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
- HandleException Function 333
- TThreadEx Class 225

8.53.1.2 Execute

Defines thread's execution.

Unit

EBase 52

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.

Important!

You **must** call the inherited implementation in your overridden `Execute` method.

A thread executes when Create 228 is called if `CreateSuspended` set to `False`, or when Start is first called after the thread is created if `CreateSuspended` set to `True`.

Examples

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.
      // 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;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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
- HandleException Function 333
- TThreadEx Class 225

8.53.2 Properties

AutoHandleException 232	No description available

See also

- TThreadEx 225
- EBase 52

8.53.2.1 AutoHandleException

Allows you to enable automatic thread's exception handling.

Unit

EBase 52

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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   // 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)

```
type
   TForm1 = class(TForm)
      ...
   private
      procedure HandleExceptionInThread(Sender: TObject);
   end;

procedure TForm1.Button1Click(Sender: TObject);
var
   Thread: TThreadEx;
begin
   Thread := TThreadEx.Create(True);
   Thread.OnTerminate := Self.HandleExceptionInThread;
   Thread.FreeOnTerminate := True;
   Thread.Start;
   Thread := nil; // never access thread var with FreeOnTerminate after
end;

procedure TForm1.HandleExceptionInThread(Sender: TObject);
var
   Thread: TThread;
begin
   Thread := (Sender as TThread);
   if Assigned(Thread.FatalException) then
      HandleException(Thread.FatalException);
end;
```

When `AutoHandleException` property is set to `True` - EurekaLog will call HandleException 333 function automatically for thread's fatal exceptions.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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
- HandleException Function 333
- TThreadEx Class 225

8.54 TWCTFreezeDetectionThread

Thread class to detects deadlocks between threads.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
TWCTFreezeDetectionThread = class(TFreezeThread 217);
```

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).

8.55 TWebDialog

Enter topic text here.

8.56 TWebDialog

Enter topic text here.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.57 TWebDialog

Enter topic text here.

8.58 TWERDialog

Enter topic text here.

8.59 TXMLLogBuilder

XML EurekaLog 7+ log builder.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
TXMLLogBuilder = class(TBaseLogBuilder 132);
```

Remarks

This is log builder class for EurekaLog 7 and above. It is used to save XML bug reports.

8.60 TZipFile

Class for creating and reading .ZIP files.

Unit

EZip 124

Syntax

Code (Delphi)

```
TZipFile = class(TObject);
```

Remarks

Methods 236	Methods
Properties 243	Properties
private	Private members
public	Public members

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.60.1 Methods

ExtractZipFile 236	Extract a ZipFile
IsValid 237	Verifies .ZIP file
ZipDirectoryContents 237	Zip the contents of a directory
Create 238	Create a TZipFile
Destroy 238	No description available
Add 238	Add a file to the ZIP file
Close 240	No description available
Extract 240	Extract a single file
ExtractAll 241	Extract All files
IndexOf 241	Translate from FileName to index in ZIP Central Header
Open 242	Opens a ZIP file for reading or writing.
Read 242	Read a file from arcive to an array of Bytes

See also

- TZipFile 235
- EZip 124

8.60.1.1 ExtractZipFile

Extract a ZipFile

Unit

EZip 124

Syntax

Code (Delphi)

```
public
class function ExtractZipFile(
   const ZipFileName, Path: String;
   const APassword: String = ''
): String; static;
```

Parameters

ZipFileName

File name of the ZIP file

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Path
Path to extract to disk

See also
- TZipFile Class 235

8.60.1.2 IsValid

Verifies .ZIP file

Unit
EZip 124

Syntax

Code (Delphi)

```
public
class function IsValid(
   const ZipFileName: String
): Boolean; static;
```

Parameters
ZipFileName
Path to Zip File

Return value
Is the .ZIP file valid

See also
- TZipFile Class 235

8.60.1.3 ZipDirectoryContents

Zip the contents of a directory

Unit
EZip 124

Syntax

Code (Delphi)

```
public
class procedure ZipDirectoryContents(
   const ZipFileName, Path: String;
   Compression: TZipCompression 570 = zcDeflate;
   const APassword: String = ''
); static;
```

Parameters
ZipFileName
File name of the ZIP file

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Path
Path of directory to zip

Compression
Compression mode.

See also
- TZipFile Class 235

8.60.1.4 Create

Create a TZipFile

Unit
EZip 124

Syntax

Code (Delphi)
```
public
constructor Create;
```

See also
- TZipFile Class 235

8.60.1.5 Destroy

Unit
EZip 124

Syntax

Code (Delphi)
```
public
destructor Destroy;
   override;
```

Remarks
Destroy will close an open zipfile before disposing of it

See also
- TZipFile Class 235

8.60.1.6 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 124

Syntax

Code (Delphi)

```
public
procedure Add(
   const FileName: string;
   ArchiveFileName: string = '';
   Compression: TZipCompression 570 = zcDeflate
); overload;

procedure Add(
   Data: TBytes;
   ArchiveFileName: string;
   Compression: TZipCompression 570 = zcDeflate
); overload;

procedure Add(
   Data: TStream;
   ArchiveFileName: string;
   Compression: TZipCompression 570 = zcDeflate
); overload;

procedure Add(
   Data: TStream;
   LocalHeader: TZipHeader 577;
   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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 235

8.60.1.7 Close

Unit
EZip 124

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 235

8.60.1.8 Extract

Extract a single file

Unit
EZip 124

Syntax

Code (Delphi)
```
public
procedure Extract(
   const FileName: string;
   const Path: string = '';
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   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 235

8.60.1.9 ExtractAll

Extract All files

Unit

EZip 124

Syntax

Code (Delphi)

```
public
procedure ExtractAll(
   const Path: string = ''
);
```

Parameters

Path
Path to extract to.

See also

- TZipFile Class 235

8.60.1.10 IndexOf

Translate from FileName to index in ZIP Central Header

Unit

EZip 124

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 235

8.60.1.11 Open

Opens a ZIP file for reading or writing.

Unit

EZip 124

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 235

8.60.1.12 Read

Read a file from arcive to an array of Bytes

Get a stream to read a file from disk

Unit

EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
public
procedure Read(
   const FileName: string;
   out Bytes: TBytes
); overload;

procedure Read(
   const FileName: string;
   out Stream: TStream;
   out LocalHeader: TZipHeader 577
); overload;
```

Parameters

FileName
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 235

8.60.2 Properties

Comment 244	No description available
FileComment 244	No description available
FileCount 244	No description available
FileInfo 245	No description available

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

FileInfos 245	No description available
FileName 245	No description available
FileNames 245	No description available
Mode 245	No description available

See also

- TZipFile 235
- EZip 124

8.60.2.1 Comment

Unit

EZip 124

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 235

8.60.2.2 FileComment

Unit

EZip 124

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 235

8.60.2.3 FileCount

Unit

EZip 124

Return value

Total files in ZIP File

See also

- TZipFile Class 235

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

8.60.2.4 FileInfo

Unit
EZip 124

Return value
TZipHeader of a File in the ZipFile

See also
- TZipFile Class 235

8.60.2.5 FileInfos

Unit
EZip 124

Return value
An array of the TZipHeader of the files in the ZIP file

See also
- TZipFile Class 235

8.60.2.6 FileName

Unit
EZip 124

Return value
FileName of a File in the ZipFile

See also
- TZipFile Class 235

8.60.2.7 FileNames

Unit
EZip 124

Return value
An array of FileNames in the ZIP file

See also
- TZipFile Class 235

8.60.2.8 Mode

Unit
EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Return value

The mode the TZipFile is opened to

See also

- TZipFile Class 235

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



9 All Functions

AllocatePageAlign 260	No description available
AllocMemCountEx 261	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemCountMax 262	Represents the maximum number of allocated memory blocks.
AllocMemSizeEx 262	Represents the total size of allocated memory blocks.
AllocMemSizeMax 263	Represents the maximum size of allocated memory.
AquireThreadData 264	Returns thread information (any thread).
BeginThreadEx 264	Spawns a separate thread of execution.
BuildBugReport 267	Builds bug report from exception information.
CheckELMagicCode 268	No description available
CheckHeap 268	Checks heap for consistency.
ConfigDelete 269	Deletes a key from the specified section.
ConfigFileName 270	Returns .ini-file name to store EurekaLog configuration.
ConfigIsRegistry 271	Indicates if EurekaLog configuration is stored in the registry or in .ini-file.
ConfigKeyPath 272	Returns registry key name to store EurekaLog configuration.
ConfigReadBool 273	Reads boolean value from specified key and section.
ConfigReadFloat 274	Reads float value from specified key and section.
ConfigReadInteger 275	Reads integer value from specified key and section.
ConfigReadString 276	Reads string value from specified key and section.
ConfigWriteBool 278	Writes boolean value to specified key

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	and section.
ConfigWriteFloat 279	Writes float value to specified key and section.
ConfigWriteInteger 280	Writes integer value to specified key and section.
ConfigWriteString 281	Writes string value to specified key and section.
ConvertAddress 282	Convert address to code to real address of code.
CRC16Hash 283	Calculates CRC-16 checksum.
CRC32Hash 284	Calculates CRC-32 checksum.
CurrentEurekaLogOptions 8	No description available
CurrentInstance 286	Returns current instance.
DecodeConfigValue 286	Low-level routine to decode data after reading from configuration storage.
DecodeInstruction 287	Decodes a single CPU instruction.
DisableMemLeaksRegistering 288	Disables memory leaks checks.
DisasmFunction 289	Disasms function.
DisasmInstruction 290	Disassebles a single CPU instruction.
DoneCheckFreeze 291	Stops background freeze detection thread.
DumpAllocationsToFile 292	Logs all allocated memory to file.
EnableMemLeaksRegistering 294	Enables memory leaks checks.
EncodeConfigValue 295	Low-level routine to encode data before saving to configuration storage.
EncryptBufferFree 295	Releases the encryption buffer.
EurekaAllocMem 297	AllocMem allocates a memory block and initializes each byte to zero.
EurekaFreeMem 298	Frees a memory block previously allocated with EurekaGetMem 298, EurekaTryGetMem 303 or

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	EurekaAllocMem 297
EurekaGetMem 298	Allocates a memory block via EurekaLog memory filter.
EurekaReallocMem 299	Resizes a memory block.
EurekaRegisterExpectedMemoryLeak 300	Register an expected memory leak.
EurekaRegisterExpectedMemoryLeakDynArray 301	Register an expected memory leak (variant for dynamic arrays).
EurekaRegisterExpectedMemoryLeakStr 302	Register an expected memory leak (variant for strings).
EurekaTryGetMem 303	Allocates a memory block via EurekaLog memory filter (w/o error checking).
EurekaUnregisterExpectedMemoryLeak 303	Removes memory block from excluded leaks.
EurekaUnregisterExpectedMemoryLeakDynArray 304	Removes memory block from excluded leaks (variant for dynamic arrays).
EurekaUnregisterExpectedMemoryLeakStr 305	Removes memory block from excluded leaks (variant for strings).
ExceptionManager 305	Exception manager.
ExpandEnvVars 306	Expands environment-variable strings.
FindDialogClass 307	Searches dialog class by name.
FindLowAddress 308	Tool function to obtain starting address to disassemble code block around known address.
GetActiveFormClass 308	Gets window class of active window.
GetActiveFormText 309	Gets caption of active window.
GetApplicationDescription 310	Gets application description.
GetApplicationName 310	Gets application name.
GetApplicationRect 311	Gets bounding rectangle for GUI application.
GetApplicationWnd 312	Returns application window (if applicable).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

GetAsmSize 312	Returns size of a single machine opcode.
GetCompanyName 313	Gets company name.
GetComputerName 314	Gets computer name user name.
GetControlClass 314	Gets class of active control.
GetControlText 315	Gets text of active control.
GetCurrentModuleCompilationDate 316	Gets compilation time of current module.
GetCurrentUserPrivileges 316	Gets description of user priviledges.
GetDiskInfo 317	Gets total and free disk space size.
GetDisplayDPI 317	Gets screen DPI.
GetDisplayMode 318	Gets screen mode.
GetEnvVariable 318	Expands single environment variable.
GetEurekaLogLabel 319	Gets full EurekaLog version.
GetEurekaLogLicense 320	Gets EurekaLog license.
GetEurekaLogPath 321	Obsolete function, do not use.
GetEurekaLogStatus 322	Gets EurekaLog stable/beta information.
GetEurekaLogVersion 322	Gets EurekaLog version.
GetExceptionModuleName 323	No description available
GetExceptionModuleVersion 324	No description available
GetFileIcon 324	No description available
GetFileSize 325	Gets file size.
GetFunctionSize 326	Returns size of the function in bytes.
GetMainWnd 327	Returns main window of application (if applicable).
GetModuleType 327	Returns type of loaded executable module, based on analyzing its file header.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

GetMXServerFromEmail 328	Retrieves mail server for e-mail address.
GetReadableSize 329	Verifies that the caller has specified access to the specified range of memory.
GetTaskBarWnd 330	Returns window of application, which creates a taskbar button (if applicable).
GetTracer 331	Get stack tracing class by number.
GetTracerClass 332	Get stack tracing class by number.
GUIFont 333	Returns font for GUI applications.
HandleException 333	Handles exception.
HookWin32API 334	Adds a function to resource monitoring.
IfElse 335	A replacement for ternary `if` operator.
IniReadString 336	Reads string from .ini file.
InitCheckFreeze 337	Starts checking for freezes and hangs.
InitSalt 338	Generates random salt.
InitWER 340	Initializes EWER unit.
IniWriteString 340	Writes string to .ini file.
InstructionToString 341	Converts decoded CPU instruction to string represenstation.
IntoIDE 342	No description available
IsAppType 342	Checks if application is of specified type.
IsBPL 343	Checks if specified module is BPL file.
IsCGI 344	Checks if application is CGI application.
IsCGIConsole 345	Checks if application is CGI console application.
IsCGIWin 345	Checks if application is CGI Windows application.
IsCLX 346	Checks if application is CLX application.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

IsCompiledWithPackages 347	Checks if this module is compiled with packages.
IsConsole 347	Checks if application is console application.
IsCPL 347	Checks if application is control panel application.
IsDebugged 348	Determines whether the calling process is being debugged by a user-mode debugger.
IsDLL 349	Checks if specified module is DLL file.
IsEurekaLogActive 349	Indicates if EurekaLog is active or disabled in this application.
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.
IsEurekaLogInstalled 354	Indicates if EurekaLog is installed in this application.
IsEXE 355	Checks if specified module is EXE file.
IsFMX 355	Checks if application is FMX application.
IsGUI 356	Checks if application is GUI application.
IsIntraWeb 357	Checks if application is IntraWeb application.
IsISAPI 357	Checks if application is ISAPI application.
IsMemLeaksEnabled 358	Indicate if memory leaks checks are enabled.
IsMemLeaksInstalled 358	No description available
IsModuleLibrary 359	Checks if specified module is DLL or BPL file.
IsReadableFile 360	Verifies that the caller has read access to the specified file.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

IsRTLCompiledWithStackFrames 360	Determines whether the RTL was compiled with Stack Frames option enabled.
IsService 361	Checks if application is a win32 service application.
IsStackFramesProc 361	Determines whether the specified function has a stack frame.
IsValidBlockAddr 362	Verifies that the caller has read access to the specified range of memory.
IsValidClass 363	Verifies that the specified pointer is a valid class.
IsValidCodeAddr 364	Verifies that the caller has execute access to the specified range of memory.
IsValidHandle 365	Verifies that this is a valid kernel handle.
IsValidObject 365	Verifies that the specified pointer is a valid object.
IsVCL 366	Checks if application is VCL application.
IsWeb 367	Checks if application is web application.
IsWin10 368	Detect Windows 10 (minimum).
IsWin64 368	Detect OS bitness.
IsWin7 369	Detect Windows 7 (minimum).
IsWin8 369	Detect Windows 8 (minimum).
IsWin81 370	Detect Windows 8.1 (minimum).
IsWinVista 371	Detect Windows Vista (minimum).
IsWinXP 371	Detect Windows XP (minimum).
IsWorkstation 372	Gets server/workstation status of current computer.
IsWritableFile 373	Verifies that the caller has write access to the specified file.
LastBugID 9	Returns BugID of last exception in current thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

MainThreadTopOfStack 374	Returns top of the stack for the main thread.
MD5Hash 374	Calculates MD5 hash.
MemLeaksName 375	Names a memory block.
MemLeaksNameDynArray 376	Names a memory block (variant for dynamic arrays).
MemLeaksNameStr 376	Names a memory block (variant for strings).
MemLeaksOwn 377	Sets ownership for memory block.
MemLeaksOwnDynArrayBoth 378	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayChild 378	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnDynArrayParent 379	Sets ownership for memory block (variant for dynamic arrays).
MemLeaksOwnStr 380	Sets ownership for memory block (variant for strings).
MemLeaksSetParentBlock 380	No description available
MemLeaksSetParentBlockDynArrayBoth 381	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayChild 381	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockDynArrayParent 382	Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).
MemLeaksSetParentBlockStr 382	Sets parent-child relation between two memory blocks for grouping (variant for strings).
MemProtect 383	Encrypts process memory.
MemUnprotect 384	Decrypts process memory.
MonoFont 386	Returns fixed-width font.
NameThread 386	Alias for NameThreadForDebugging to

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	avoid conflicts with System.Classes.TThread.NameThreadForDebugging.
NameThreadForDebugging 387	Sets the name for a thread.
PageSize 388	No description available
PauseFreezeCheck 388	Temporarily disables freeze checks.
PrimaryInstance 389	The same as MainInstance. See remarks for more info.
RaiseExpected 389	Raises expected exception.
RaiseFreezeException 390	Raises freeze/hang/deadlock exception for the specified thread.
RegisterDialogClass 391	Registers dialog class.
RegisterDialogClassFirst 391	Registers dialog class.
RegisterEventAttachedFilesRequest 392	Registers OnAttachedFilesRequest event handler.
RegisterEventBeginReportGen 393	Registers OnBeginReportGen event handler.
RegisterEventCustomBugID 394	Registers OnCustomBugID event handler.
RegisterEventCustomButtonClick 395	Registers OnCustomButtonClick event handler.
RegisterEventCustomDataRequest 396	Registers OnCustomDataRequest event handler.
RegisterEventCustomFileName 397	Registers OnCustomFileName event handler.
RegisterEventCustomWebFieldsRequest 398	Registers OnCustomWebFieldRequest event handler.
RegisterEventEndReportGen 399	Registers OnEndReportGen event handler.
RegisterEventExceptionAction 400	Registers OnExceptionAction event handler.
RegisterEventExceptionError 401	Registers OnExceptionError event handler.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

RegisterEventExceptionNotify 402	Registers OnExceptionNotify event handler.
RegisterEventHelpButtonClick 403	Registers OnHelpButtonClick event handler.
RegisterEventPasswordRequest 404	Registers OnPasswordRequest event handler.
RegisterEventRaise 405	Registers OnRaise event handler.
RegisterEventReproduceRequest 406	Registers OnReproduceRequest event handler.
RegisterEventZippedFilesRequest 407	Registers OnZippedFilesRequest event handler.
RegKeyCheck 408	Checks registry key for existing.
RegKeyRead 408	Reads string from registry key.
RegKeyWrite 409	Writes string to registry key.
ResumeFreezeCheck 410	Enables freeze checks back.
RSADecrypt 411	Decrypts encrypted data by using RSA cipher.
RSADestroyKey 414	Deletes RSA key.
RSAEncrypt 414	Encrypts "clear text" data by using RSA cipher.
RSAGenKey 417	Creates new RSA key.
RSALoadPrivateKey 418	Imports private key.
RSALoadPublicKey 419	Imports public key.
RSASavePrivateKey 420	Exports private key.
RSASavePublicKey 421	Exports public key.
RSASign 422	Signs data.
RSAVerify 424	Verifies digital signature.
SafeExec 426	Executes routine with exception handling. Any exception is handled as fatal exception.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

SDBMHash 427	Calculates SDBM hash.
SecureFree 428	Wipes and releases memory.
SetAppType 429	Set a type of this application.
SetEurekaLogState 6	Enables or disables EurekaLog in this application.
SetEurekaLogState 431	Enables or disables EurekaLog in this application.
SetEurekaLogStateInThread 7	Enables or disables EurekaLog in the specified thread.
SetupConfig 433	Changes EurekaLog configuration storage.
SHA1Hash 434	Calculates SHA-1 hash.
SHA256Hash 434	Calculates SHA-2/SHA-256 hash.
ShowExpected 435	Shows expected exception.
SignatureGetInfo 436	Returns information about digital signature and its certificate of an executable file.
TEADecrypt 437	Decrypts encrypted data by using TEA cipher.
TEADefaultKey 441	Returns pre-initialized default key for TEA.
TEADeriveKey 443	Initializes TEA key from password.
TEADestroyKey 447	Fills TEA key with zeros.
TEAEncrypt 447	Encrypts "clear text" data by using TEA cipher.
TEAInitSessionKey 451	Generates random TEA key.
ThreadData 452	Returns thread information (current thread only).
ThreadDataEnum 453	Enumerates running threads.
TwofishDecrypt 454	Decrypts encrypted data by using Twofish cipher.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TwofishDeriveKey 455	Initializes Twofish key from password.
TwofishDestroyKey 457	Deletes Twofish key.
TwofishEncrypt 457	Encrypts "clear text" data by using Twofish cipher.
TwofishInitIV 459	Creates new IV.
TwofishInitKey 459	"Imports" (precalculates) Twofish key from RAW key value.
TZipCompressionToString 460	Converts ZIP compression method value to string representation.
UnRegisterEventAttachedFilesRequest 461	Unregisters previously registered OnAttachedFilesRequest event handler.
UnRegisterEventBeginReportGen 462	Unregisters previously registered OnBeginReportGen event handler.
UnRegisterEventCustomBugID 462	Unregisters previously registered OnCustomBugID event handler.
UnRegisterEventCustomButtonClick 463	Unregisters previously registered OnCustomButtonClick event handler.
UnRegisterEventCustomDataRequest 464	Unregisters previously registered OnCustomDataRequest event handler.
UnRegisterEventCustomFileName 465	Unregisters previously registered OnCustomFileName event handler.
UnRegisterEventCustomWebFieldsRequest 466	Unregisters previously registered OnCustomWebFieldRequest event handler.
UnRegisterEventEndReportGen 466	Unregisters previously registered OnEndReportGen event handler.
UnRegisterEventExceptionAction 467	Unregisters previously registered OnExceptionAction event handler.
UnRegisterEventExceptionError 468	Unregisters previously registered OnExceptionError event handler.
UnRegisterEventExceptionNotify 469	Unregisters previously registered OnExceptionNotify event handler.
UnRegisterEventHelpButtonClick 470	Unregisters previously registered OnHelpButtonClick event handler.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

UnRegisterEventPasswordRequest 470	Unregisters previously registered OnPasswordRequest event handler.
UnRegisterEventRaise 471	Unregisters previously registered OnRaise event handler.
UnRegisterEventReproduceRequest 472	Unregisters previously registered OnReproduceRequest event handler.
UnRegisterEventZippedFilesRequest 473	Unregisters previously registered OnZippedFilesRequest event handler.
WCTDone 474	Finalize EWCT unit.
WCTInit 474	Initialize EWCT unit.

9.1 AllocatePageAlign

Allocates memory with alignment on current memory's page size boundary.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function AllocatePageAlign(
   const ASize: Cardinal;
   out AHolder: Pointer
): Pointer;
```

Parameters

ASize

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 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 388.

See also

- AllocMem
- GetMem
- FreeMem
- PageSize 388

9.2 AllocMemCountEx

Represents the total number of allocated memory blocks in a Delphi application.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function AllocMemCountEx: Int64;
```

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 262

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- AllocMemSizeEx Function 262
- IsMemLeaksInstalled Function 358
- IsMemLeaksEnabled Function 358
- EurekaGetMem Function 298

9.3 AllocMemCountMax

Represents the maximum number of allocated memory blocks.

Unit

EMemLeaks 96

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 261.

Remarks

`AllocMemCountMax` returns maximum of AllocMemCountEx 261. 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 261
- AllocMemSizeMax Function 263
- IsMemLeaksInstalled Function 358
- IsMemLeaksEnabled Function 358
- EurekaGetMem Function 298

9.4 AllocMemSizeEx

Represents the total size of allocated memory blocks.

Unit

EMemLeaks 96

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 263
- AllocMemCountEx Function 261
- IsMemLeaksInstalled Function 358
- IsMemLeaksEnabled Function 358
- EurekaGetMem Function 298

9.5 AllocMemSizeMax

Represents the maximum size of allocated memory.

Unit

EMemLeaks 96

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 filter is not installed, or no allocations were made. This value is always equals or greater than AllocMemSizeEx 262.

Remarks

`AllocMemSizeMax` returns maximum of AllocMemSizeEx 262. 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 262
- AllocMemCountMax Function 262
- IsMemLeaksInstalled Function 358
- IsMemLeaksEnabled Function 358
- EurekaGetMem Function 298

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.6 AquireThreadData

Returns thread information (any thread).

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
function AquireThreadData(
   const AThreadID: Cardinal;
   const AForce: Boolean = False
): TEurekaThreadData 190;
```

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 190 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 452 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 452
- ThreadDataEnum Function 453
- TEurekaThreadData Class 190

9.7 BeginThreadEx

Spawns a separate thread of execution.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EBase 52

Syntax

Code (Delphi)

```
function BeginThreadEx(
   SecurityAttributes: Pointer;
   StackSize: LongWord;
   ThreadFunc: TThreadFunc;
   Parameter: Pointer;
   CreationFlags: LongWord;
   var ThreadId: TThreadID;
   const AThreadName: String = '';
   const AEnableEL: Boolean = True
): 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 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. Default is True. This argument is ignored if you compile your application without EurekaLog.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 225 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 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

- TThreadEx Class 225
- SetEurekaLogStateInThread Function
- NameThread Function
- HandleException Function 333

9.8 BuildBugReport

Builds bug report from exception information.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
function BuildBugReport(
   const AExceptionInfo: TEurekaExceptionInfo 157
): String;
```

Parameters

AExceptionInfo [in]

Information about exception to build bug report.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Return value

Textual representation of bug report. Exact value depends on used log builder class. Default log builder class is TLogBuilder 223.

Remarks

This function builds bug report for the specified exception information by using default 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.

9.9 CheckELMagicCode

Checks double word for being one of the known EurekaLog's "magic codes" (signatures).

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function CheckELMagicCode(
   const ACode: DWord
): Boolean;
```

Parameters

ACode

A value to check.

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 60 unit, but also used in some other places where you need to check if this is valid EurekaLog's data or not.

9.10 CheckHeap

Checks heap for consistency.

Unit

EMemLeaks 96

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 358 with enabled registering memory leaks 358.

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 292

9.11 ConfigDelete

Deletes a key from the specified section.

Unit

EConfig 57

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

This routine deletes a key from the section in EurekaLog configuration.

See EConfig 57 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 57
- RegKeyDelete
- IniWriteString 340

9.12 ConfigFileName

Returns .ini-file name to store EurekaLog configuration.

Unit

EConfig 57

Syntax

Code (Delphi)

```
function ConfigFileName: String;
```

Return value

.ini-file name.

Exceptions

System.SysUtils.EAssertionFailed	If called when ConfigIsRegistry is True.

Remarks

Warning

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 276 or ConfigWriteString 281 (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 57 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 433 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 433
- ConfigIsRegistry 271
- ConfigKeyPath 272

9.13 ConfigIsRegistry

Indicates if EurekaLog configuration is stored in the registry or in .ini-file.

Unit

EConfig 57

Syntax

Code (Delphi)

```
function ConfigIsRegistry: Boolean;
```

Return value

True - EurekaLog configuration is stored in the registry. Use ConfigKeyPath 272 to get registry key name.

False - EurekaLog configuration is stored in the .ini-file. Use ConfigFileName 270 to get .ini-file name.

Remarks

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 276 or ConfigWriteString 281 (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 57 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 433 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 433
- ConfigKeyPath 272
- ConfigFileName 270

9.14 ConfigKeyPath

Returns registry key name to store EurekaLog configuration.

Unit

EConfig 57

Syntax

Code (Delphi)

```
function ConfigKeyPath: String;
```

Return value

Registry key name with leading and trailing '\' delimiters.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 276 or ConfigWriteString 281 (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 57 unit description for basic desciption of EurekaLog configuration.

Code may change this value by calling SetupConfig 433 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 433
- ConfigIsRegistry 271
- ConfigFileName 270

9.15 ConfigReadBool

Reads boolean value from specified key and section.

Unit

EConfig 57

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 276 function.

Examples

Code (Delphi)

```
CheckUpdates.Checked := ConfigReadBool(IDESectionName, IDECanCheckUpdate
```

See also

- ConfigReadString 276
- ConfigReadInteger 275
- ConfigReadFloat 274

9.16 ConfigReadFloat

Reads float value from specified key and section.

Unit

EConfig 57

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 276 function.

Examples

Code (Delphi)

```
StartDate := ConfigReadFloat(Name, Key);
```

See also

- ConfigReadString 276
- ConfigReadInteger 275
- ConfigReadBool 273

9.17 ConfigReadInteger

Reads integer value from specified key and section.

Unit

EConfig 57

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigReadString 276 function.

Examples

Code (Delphi)

```
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 276
- ConfigReadBool 273
- ConfigReadFloat 274

9.18 ConfigReadString

Reads string value from specified key and section.

Unit

EConfig 57

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 57 unit description for basic desciption of EurekaLog configuration.

Examples

Code (Delphi)

```
function GetUserEmail: String;
begin
   Result := ConfigReadString(IniUserSettingsSection, IniUserSettingsEMa
end;
```

See also

- ConfigWriteString 281
- ConfigReadInteger 275
- ConfigReadBool 273
- ConfigReadFloat 274
- ConfigDelete 269
- SetupConfig 433
- RegKeyRead 408
- IniReadString 336
- DecodeConfigValue 286

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.19 ConfigWriteBool

Writes boolean value to specified key and section.

Unit

EConfig 57

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 269 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes boolean value to EurekaLog configuration.

See EConfig 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 281 function.

Examples

Code (Delphi)

```
ConfigWriteBool(IDESectionName, IDECanCheckUpdatesName, CheckUpdates.Che
```

See also

- ConfigWriteString 281
- ConfigWriteInteger 280
- ConfigWriteFloat 279
- ConfigDelete 269

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.20 ConfigWriteFloat

Writes float value to specified key and section.

Unit

EConfig 57

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 269 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes float value to EurekaLog configuration.

See EConfig 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 281 function.

Examples

Code (Delphi)

```
if StartDate = 0 then
begin
   StartDate := Now;
   ConfigWriteFloat(Name, Key, StartDate);
end;
```

See also

- ConfigWriteString 281
- ConfigWriteInteger 280
- ConfigWriteBool 278
- ConfigDelete 269

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.21 ConfigWriteInteger

Writes integer value to specified key and section.

Unit

EConfig 57

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 269 procedure instead.

Exceptions

System.SysUtils.EOSError	Raises this exception on failure.

Remarks

Writes integer value to EurekaLog configuration.

See EConfig 57 unit description for basic desciption of EurekaLog configuration.

This function is a wrapper for ConfigWriteString 281 function.

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 281
- ConfigWriteBool 278
- ConfigWriteFloat 279

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ConfigDelete 269

9.22 ConfigWriteString

Writes string value to specified key and section.

Unit

EConfig 57

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 269 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.

Remarks

Writes string value to EurekaLog configuration.

See EConfig 57 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 276
- ConfigWriteInteger 280
- ConfigWriteBool 278

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- ConfigWriteFloat 279
- ConfigDelete 269
- SetupConfig 433
- RegKeyWrite 409
- IniWriteString 340
- EncodeConfigValue 295

9.23 ConvertAddress

Convert address to code to real address of code.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function ConvertAddress(
   const AAddr: Pointer
): Pointer;
```

Parameters

AAddr

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.

- 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

- IsValidBlockAddr 362
- IsValidCodeAddr 364

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.24 CRC16Hash

Calculates CRC-16 checksum.

Unit

EHash 87

Syntax

Code (Delphi)

```
function CRC16Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TCRC16Hash 491; overload;

function CRC16Hash(
   const AData: RawByteString
): TCRC16Hash 491; 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.

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 284

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.25 CRC32Hash

Calculates CRC-32 checksum.

Unit

EHash 87

Syntax

Code (Delphi)

```
function CRC32Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TCRC32Hash 492; overload;

function CRC32Hash(
   const AData: RawByteString
): TCRC32Hash 492; 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 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 283

9.26 CurrentEurekaLogOptions

Returns current EurekaLog's options.

Unit

ExceptionLog7 121

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function CurrentEurekaLogOptions: TEurekaModuleOptions 188;
```

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 188 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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
if ApplicationRunOnDeveloperMachine then
   CurrentEurekaLogOptions.SenderClasses := '';
```

9.27 CurrentInstance

Returns current instance.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function CurrentInstance: HMODULE;
```

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.

Examples

Code (Delphi)

```
Options := EurekaLogOptions(CurrentInstance);
```

See also

- PrimaryInstance 389
- HInstance
- MainInstance

9.28 DecodeConfigValue

Low-level routine to decode data after reading from configuration storage.

Unit

EConfig 57

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function DecodeConfigValue(
   const AValue: String
): String;
```

Parameters

AValue

Encoded data, which was retrieved from configuration storage. This data should be processed by EncodeConfigValue 295 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 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

Decodes data which was previosly encoded by EncodeConfigValue 295.

Usually you don't need to use this function, because you're encouraged to use ConfigReadString 276 or it's wrapper. These ConfigReadXYZ functions perform such decoding automatically.

See also

- EncodeConfigValue 295
- ConfigReadString 276

9.29 DecodeInstruction

Decodes a single CPU instruction.

Unit

EDisAsm 70

Syntax

Code (Delphi)

```
function DecodeInstruction(
   Address: PAddress;
   const AExternalProcess: Boolean = False
): TInstruction 537;
```

Parameters

Address [in]

A pointer to CPU instruction to decode.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.).

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 341 function.

Note
You can use DisasmInstruction 290 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 290
- InstructionToString function 341
- DisasmFunction function 289

9.30 DisableMemLeaksRegistering

Disables memory leaks checks.

Unit

EMemLeaks 96

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 358. You can enable regestering memory leaks back with EnableMemLeaksRegistering function 294.

- 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 96.

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 300 to register an expected memory leak or to

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

disable leak reporting for already allocated memory block.

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

- EnableMemLeaksRegistering Function 294
- IsMemLeaksEnabled Function 358
- ActivateMemoryLeakCheck Function
- EurekaRegisterExpectedMemoryLeak Function 300

9.31 DisasmFunction

Disasms function.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
function DisasmFunction(
   const Address: Pointer;
   const AExceptionAddress: Pointer = nil;
   const ASize: Integer = 0;
   const ALines: Integer = 0;
   const AOptions: TEurekaModuleOptions 188 = 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Return value

A textual representation of disassembled function including comments, marks, addresses, RAW bytes and decoded instructions.

Remarks

This function is used to generate disasm view in bug reports. This function uses EDisAsm 70 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 308 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 308, it's recommended that FindLowAddress's `ALines` argument equals to DisasmFunction's `ALines` argument.

See also

- FindLowAddress function 308
- DisasmInstruction function 290
- DisasmInstruction function 326

9.32 DisasmInstruction

Disassebles a single CPU instruction.

Unit

EDisAsm 70

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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:

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 287 and InstructionToString 341. This function is used as base for DisasmFunction 289 function.

See also

- DecodeInstruction function 287
- InstructionToString function 341
- DisasmFunction function 289

9.33 DoneCheckFreeze

Stops background freeze detection thread.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
procedure DoneCheckFreeze;
```

Remarks

This function will stop running background freeze detection thread launched by InitCheckFreeze 337. This function will do nothing if no such thread is running.

Use this function to completely stop freeze detection.

Tip

Use PauseFreezeCheck 388 to temporarily turn off freeze detection - without full stop/re-initialization.

See also

- InitCheckFreeze Function 337

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- PauseFreezeCheck Function 388

9.34 DumpAllocationsToFile

Logs all allocated memory to file.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure DumpAllocationsToFile(
   const AFileName: String;
   const ACount: Boolean = True;
   const AGroup: Boolean = True;
   const ASort: Boolean = True;
   const AFilter: TDumpFilterFunc 493 = nil;
   const ACompare: TDumpCompareFunc 493 = nil;
   const ACompareForCount: TDumpCompareFunc 493 = 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.

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 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 380 or MemLeaksOwn function 377. Additionally, any memory allocated withing constructor of any object will be declared as child memory to this object.

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.

You can use MemLeaksName 375 or MemLeaksOwn 377 functions to mark any memory block with arbitrary name tag. Name tag will be displayed in header for memory block in the report.

Tip

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 358 with enabled registering memory leaks 358.

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 268
- TDumpFilterFunc Type 493
- TDumpCompareFunc Type 493

9.35 EnableMemLeaksRegistering

Enables memory leaks checks.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure EnableMemLeaksRegistering;
```

Remarks

Enables back registering of memory leaks which was previously disabled with DisableMemLeaksRegistering 288.

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- DisableMemLeaksRegistering Function 288
- IsMemLeaksEnabled Function 358

9.36 EncodeConfigValue

Low-level routine to encode data before saving to configuration storage.

Unit

EConfig 57

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 276 or ConfigWriteString 281 (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 281 or it's wrapper. These ConfigWriteXYZ functions perform such encoding automatically.

See also

- DecodeConfigValue 286
- ConfigWriteString 281

9.37 EncryptBufferFree

Releases the encryption buffer.

Unit

EEncrypt 73

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure EncryptBufferFree(
   var AData: TEncryptBuffer 525
);
```

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 73 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)

```
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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   // (we did not allocate memory for buffer)
end;
```

See also

- EncryptBufferFree Function 295

9.38 EurekaAllocMem

AllocMem allocates a memory block and initializes each byte to zero.

Unit

EMemLeaks 96

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 298. If there is not enough memory available to allocate the block, an IeOutOfMemory 540 error is raised.

Note

If the memory does not need to be zero-initialized, you can use EurekaGetMem 298.

Note

Do not call `EurekaAllocMem` manually. This function will be called automatically when you call AllocMem function when EurekaLog's memory filter is installed.

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 303 function to allocate memory with ignoring errors.

See also

- EurekaGetMem Function 298
- EurekaTryGetMem Function 303

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EurekaFreeMem Function 298
- EurekaReallocMem Function 299
- IsMemLeaksEnabled Function 358
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

9.39 EurekaFreeMem

Frees a memory block previously allocated with EurekaGetMem 298, EurekaTryGetMem 303 or EurekaAllocMem 297.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaFreeMem(
    P: Pointer
): Integer;
```

Parameters

P [in]

A pointer to memory block returned by EurekaGetMem 298, EurekaTryGetMem 303 or EurekaAllocMem 297.

Return value

Return value is always 0.

Remarks

Use this procedure to dispose of a memory block obtained with EurekaGetMem 298, EurekaTryGetMem 303 or EurekaAllocMem 297.

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 298
- EurekaTryGetMem Function 303
- EurekaAllocMem Function 297
- EurekaReallocMem Function 299

9.40 EurekaGetMem

Allocates a memory block via EurekaLog memory filter.

Unit

EMemLeaks 96

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 298. If there is not enough memory available to allocate the block, an IeOutOfMemory 540 error is raised.

Note

If the memory block must be initialized with zero, you can use EurekaAllocMem 297.

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 303 function to allocate memory with ignoring errors.

See also

- EurekaTryGetMem Function 303
- EurekaAllocMem Function 297
- EurekaFreeMem Function 298
- EurekaReallocMem Function 299
- IsMemLeaksEnabled Function 358
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

9.41 EurekaReallocMem

Resizes a memory block.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
function EurekaReallocMem(
   P: Pointer;
   Size: PtrInt
): Pointer;
```

Parameters

P [in]

A pointer to memory block returned by EurekaGetMem 298, EurekaTryGetMem 303 or EurekaAllocMem 297.

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` cannot 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 303 function to allocate memory with ignoring errors.

See also

- EurekaGetMem Function 298
- EurekaTryGetMem Function 303
- EurekaAllocMem Function 297
- EurekaFreeMem Function 298
- IsMemLeaksEnabled Function 358
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

9.42 EurekaRegisterExpectedMemoryLeak

Register an expected memory leak.

Unit

EMemLeaks 96

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 288.

Tip

While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeak 303 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 301 and EurekaRegisterExpectedMemoryLeakStr 302 for dynamic arrays and strings instead of this function.

See also

- EurekaUnregisterExpectedMemoryLeak Function 303
- EurekaRegisterExpectedMemoryLeakDynArray Function 301
- EurekaRegisterExpectedMemoryLeakStr Function 302
- DisableMemLeaksRegistering Function 288

9.43 EurekaRegisterExpectedMemoryLeakDynArray

Register an expected memory leak (variant for dynamic arrays).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaRegisterExpectedMemoryLeakDynArray(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
    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 300, but it accepts only pointers to dynamic arrays.

Tip

While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeakDynArray 304 function to unregister memory block from excluded leaks.

See also

- EurekaUnregisterExpectedMemoryLeakDynArray Function 304
- EurekaRegisterExpectedMemoryLeak Function 300

9.44 EurekaRegisterExpectedMemoryLeakStr

Register an expected memory leak (variant for strings).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaRegisterExpectedMemoryLeakStr(
    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 300, but it accepts only pointers to strings (ANSI or Unicode, but not WideStrings).

Tip

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

While it is usually not needed, but you can call EurekaUnregisterExpectedMemoryLeakStr 305 function to unregister memory block from excluded leaks.

See also

- EurekaUnregisterExpectedMemoryLeakStr Function 305
- EurekaRegisterExpectedMemoryLeak Function 300

9.45 EurekaTryGetMem

Allocates a memory block via EurekaLog memory filter (w/o error checking).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaTryGetMem(
   Size: PtrUInt
): 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 298. If there is not enough memory available to allocate the block, `nil` is returned.

Note

Unlike EurekaGetMem 298, `EurekaTryGetMem` function can be called manually.

See also

- EurekaGetMem Function 298
- EurekaAllocMem Function 297
- EurekaFreeMem Function 298
- EurekaReallocMem Function 299
- IsMemLeaksEnabled Function 358
- MemLeaksError Event Handler
- MemLeaksErrorsToIgnore Global Variable

9.46 EurekaUnregisterExpectedMemoryLeak

Removes memory block from excluded leaks.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeak(
    P: Pointer
): Boolean;
```

Parameters

P [in]

A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeak function 300

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 300 returned `False`).

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeak Function 300

9.47 EurekaUnregisterExpectedMemoryLeakDynArray

Removes memory block from excluded leaks (variant for dynamic arrays).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeakDynArray(
    P: Pointer
): Boolean;
```

Parameters

P [in]

A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeakDynArray function 301

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 301 returned `False`).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeakDynArray Function 301

9.48 EurekaUnregisterExpectedMemoryLeakStr

Removes memory block from excluded leaks (variant for strings).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
function EurekaUnregisterExpectedMemoryLeakStr(
    P: Pointer
): Boolean;
```

Parameters

P [in]

A pointer to memory block which was previosly passed to EurekaRegisterExpectedMemoryLeakStr function 302.

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 302 returned `False`).

Remarks

This function removes a successfully registered expected memory leak from list of exclusions.

See also

- EurekaRegisterExpectedMemoryLeakStr Function 302

9.49 ExceptionManager

Exception manager.

Unit

EExceptionManager 84

Syntax

Code (Delphi)

```
function ExceptionManager: TExceptionManager 203;
```

Return value

Instance of exception manager.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 203 for more information.

Note
This function is the same as TExceptionManager 203. Instance 204.

See also

- TExceptionManager class 203
- TExceptionManager.Instance class method 204

9.50 ExpandEnvVars

Expands environment-variable strings.

Unit

ESysInfo 115

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

If you don't specify this argument - the default converter will be used (GetEnvVariable 318).

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

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 318
- GetEnvVariable 318

9.51 FindDialogClass

Searches dialog class by name.

Unit

EDialog 64

Syntax

Code (Delphi)

```
function FindDialogClass(
   const AClassName: String
): TBaseDialogClass 482;
```

Parameters

`AClassName` [in]

Name of dialog class. For example: 'TMessageBoxDialog'.

Return value

Registered dialog class or `nil` if no class was found.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Functions looks for registered dialog class with the specified name and return first found class. Dialog class registration is performed by RegisterDialogClass 391 and RegisterDialogClassFirst 391 functions.

See also

- RegisterDialogClass 391
- RegisterDialogClassFirst 391

9.52 FindLowAddress

Tool function to obtain starting address to disassemble code block around known address.

Unit

ELogBuilder 90

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 289. 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 289, it's recommended that FindLowAddress's ALines argument equals to DisasmFunction's ALines argument.

9.53 GetActiveFormClass

Gets window class of active window.

Unit

ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
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 309
- GetControlClass 314
- GetControlText 315

9.54 GetActiveFormText

Gets caption of active window.

Unit

ESysInfo 115

Syntax

```
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.

If the function fails, the return value is error message.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 308
- GetControlClass 314
- GetControlText 315

9.55 GetApplicationDescription

Gets application description.

Unit

ESysInfo 115

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 389)

See also

- GetModuleDescription
- GetApplicationName 310
- GetExceptionModuleName 323
- PrimaryInstance 389
- CurrentInstance 286

9.56 GetApplicationName

Gets application name.

Unit

ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 389)

See also

- GetModuleDescriptionAndVersion
- GetApplicationDescription 310
- GetExceptionModuleName 323
- PrimaryInstance 389
- CurrentInstance 286

9.57 GetApplicationRect

Gets bounding rectangle for GUI application.

Unit

ESysInfo 115

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 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

enabled - to determinate region of desktop to take screenshot from.

See also
- Screenshot routines 105

9.58 GetApplicationWnd

Returns application window (if applicable).

Unit
EAppType 50

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.

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 _GetAppWnd handler.

Note

Typically this is Application.Handle.

See also
- GetMainWnd 327
- GetTaskBarWnd 330
- _GetAppWnd
- IsAppType 342

9.59 GetAsmSize

Returns size of a single machine opcode.

Unit
EDisAsm 70

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
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 successfull. 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 282

9.60 GetCompanyName

Gets company name.

Unit

ESysInfo 115

Syntax

Code (Delphi)

```
function GetCompanyName: String;
```

Return value

Company name or empty string.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 314
- GetUserEMail
- GetUserFullName
- GetUserName

9.61 GetComputerName

Gets computer name user name.

Unit

ESysInfo 115

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 313
- GetUserEMail
- GetUserFullName
- GetUserName

9.62 GetControlClass

Gets class of active control.

Unit

ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
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 308
- GetActiveFormText 309
- GetControlText 315

9.63 GetControlText

Gets text of active control.

Unit

ESysInfo 115

Syntax

```
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 - *).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- GetActiveFormClass 308
- GetActiveFormText 309
- GetControlClass 314

9.64 GetCurrentModuleCompilationDate

Gets compilation time of current module.

Unit
ESysInfo 115

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(PrimaryInstance 389).

Please see GetCompilationDate for limitations of this function.

See also
- GetCompilationDate
- PrimaryInstance 389
- CurrentInstance 286

9.65 GetCurrentUserPrivileges

Gets description of user priviledges.

Unit
ESysInfo 115

Syntax

Code (Delphi)
```
function GetCurrentUserPrivileges: String;
```

Return value
Status of enabled and disabled system priviledges 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

SeUndockPrivilege - OFF

SeIncreaseWorkingSetPrivilege - OFF

SeTimeZonePrivilege - OFF

Remarks
Function opens current process and ask for security token, then enumerates priviledges listed in this token and collect status of all included priviledges.

9.66 GetDiskInfo

Gets total and free disk space size.

Unit
ESysInfo 115

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 286).

See also
- GetFreeDisk
- GetTotalDisk
- CurrentInstance 286
- PrimaryInstance 389

9.67 GetDisplayDPI

Gets screen DPI.

Unit
ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
function GetDisplayDPI: Integer;
```

Return value
DPI of desktop.

Example values: 96, 120, etc.

Remarks
Basically, this is GetDeviceCaps(GetDC(0), LOGPIXELSY).

See also
- GetDisplayMode 318

9.68 GetDisplayMode

Gets screen mode.

Unit
ESysInfo 115

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 317

9.69 GetEnvVariable

Expands single environment variable.

Unit
ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
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 306 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 306

9.70 GetEurekaLogLabel

Gets full EurekaLog version.

Unit

ESysInfo 115

Syntax

Code (Delphi)

```
function GetEurekaLogLabel: String;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Return value

Actual EurekaLog version information.

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 322 + GetEurekaLogLicense 320 + GetEurekaLogStatus 322.

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 584 and EEurekaLogLicense 583 constants.

See also

- GetEurekaLogVersion 322
- GetEurekaLogLicense 320
- GetEurekaLogStatus 322
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 584
- EEurekaLogCurrentVersion 583
- EEurekaLogLicense 583

9.71 GetEurekaLogLicense

Gets EurekaLog license.

Unit

ESysInfo 115

Syntax

Code (Delphi)

```
function GetEurekaLogLicense: String;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Return value

Actual EurekaLog license information.

Example values:

'Trial'

'Professional'

'Enterprise'

'Registered' (for not installed EurekaLog)

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
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 583 constant.

See also

- GetEurekaLogLabel 319
- GetEurekaLogVersion 322
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 584
- EEurekaLogCurrentVersion 583
- EEurekaLogLicense 583

9.72 GetEurekaLogPath

Obsolete function, do not use.

Unit

ESysInfo 115

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Use GetFolderEurekaLog function instead.

See also
- GetFolderEurekaLog

9.73 GetEurekaLogStatus

Gets EurekaLog stable/beta information.

Unit
ESysInfo 115

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 319
- GetEurekaLogVersion 322
- GetEurekaLogLicense 320
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 584
- EEurekaLogCurrentVersion 583
- EEurekaLogLicense 583

9.74 GetEurekaLogVersion

Gets EurekaLog version.

Unit
ESysInfo 115

Syntax

Code (Delphi)
```
function GetEurekaLogVersion: String;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 584 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
- GetEurekaLogLabel 319
- GetEurekaLogLicense 320
- GetFolderEurekaLog
- GetFolderEurekaLogAppData
- EEurekaLogVersion 584
- EEurekaLogCurrentVersion 583
- EEurekaLogLicense 583

9.75 GetExceptionModuleName

Gets description of exception's module.

Unit
ESysInfo 115

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 310.

This function is simple combination GetModuleDescription(FindHInstance(AAddr))

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- GetModuleDescription
- GetApplicationDescription 310
- GetExceptionModuleVersion 324

9.76 GetExceptionModuleVersion

Gets exception's module version.

Unit

ESysInfo 115

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 323
- GetFileVersion

9.77 GetFileIcon

Gets executable file's icon.

Unit

ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function GetFileIcon(
   const AFileName: String;
   ALarge: Boolean;
   ASz: PSize
): HICON;
```

Parameters

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. 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.

9.78 GetFileSize

Gets file size.

Unit

ESysInfo 115

Syntax

Code (Delphi)

```
function GetFileSize(
   const AFileName: String
): UInt64;
```

Parameters

AFileName [in]
Full file name.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

9.79 GetFunctionSize

Returns size of the function in bytes.

Unit
EDisAsm 70

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Warning
This is heuristic function. Results are not 100% reliable.

See also

- GetAsmSize
- ConvertAddress 282

9.80 GetMainWnd

Returns main window of application (if applicable).

Unit

EAppType 50

Syntax

Code (Delphi)

```
function GetMainWnd: HWND;
```

Return value

Handle of main 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.

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 _GetMainWnd handler.

Note
Typically this is Application.MainForm.Handle.

See also

- _GetTaskBarWnd 330
- _GetApplicationWnd 312
- _GetMainWnd
- IsAppType 342

9.81 GetModuleType

Returns type of loaded executable module, based on analyzing its file header.

Unit

EAppType 50

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
function GetModuleType(
   const AInstance: HMODULE = 0
): TModuleType;
```

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, TModuleType.mtLibrary or TModuleType.mtPackage on success and TModuleType.mtUnknown 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.

There are also simple wrappers for more easy access. For example, IsEXE, IsDLL and so on. See EAppType unit for more info.

See also

- TModuleType
- CurrentInstance

9.82 GetMXServerFromEmail

Retrieves mail server for e-mail address.

Unit

EDNS

Syntax

Code (Delphi)

```
function GetMXServerFromEmail(
   const EMail: String
): String;
```

Parameters

EMail [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 130	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.

9.83 GetReadableSize

Verifies that the caller has specified access to the specified range of memory.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function GetReadableSize(
   const AAddress: Pointer;
   const ASize: Cardinal;
   const AAttributes: Cardinal = 0
): Cardinal;
```

Parameters

AAddress

A pointer to the first byte of the memory block.

ASize

The size of the memory block, in bytes. If this parameter is zero, the return value is zero.

AAttributes

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

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 362

9.84 GetTaskBarWnd

Returns window of application, which creates a taskbar button (if applicable).

Unit

EAppType 50

Syntax

Code (Delphi)

```
function GetTaskBarWnd: HWND;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

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 _GetTaskBarWnd handler.

Note
Typically this is Application.MainForm.Handle for modern applications and Application.Handle for old applications.

See also
- GetMainWnd 327
- GetApplicationWnd 312
- _GetTaskBarWnd
- IsAppType 342

9.85 GetTracer

Get stack tracing class by number.

Unit
ECallStack 54

Syntax

Code (Delphi)

```
function GetTracer(
   const ATracerNo: Integer
): TEurekaBaseStackList 149;
```

Parameters
ATracerNo [in]
Number of stack tracer. Numbers are listed as Tracer*XYZ* constants in EStackTracing 114 unit. Some examples are: TracerRawEurekaLogV7 TracerRawEurekaLogV6 TracerRawEurekaLog TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesEurekaLogV7 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

Use this function to get call stack class for specific tracer of your choice.

Tip

Use EurekaCallStackClass 584.`Create` when you don't know which class do you need.

Tip

This function is preferred over GetTracerClass 332 function.

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

- GetTracerClass 332
- TEurekaBaseStackList.GetCurrentInstruction method 150

9.86 GetTracerClass

Get stack tracing class by number.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
function GetTracerClass(
   const ATracerNo: Integer
): TEurekaCallStackClass 531;
```

Parameters

ATracerNo [in]

Number of stack tracer. Numbers are listed as Tracer*XYZ* constants in EStackTracing 114 unit. Some examples are: TracerRawEurekaLogV7 TracerRawEurekaLogV6 TracerRawEurekaLog TracerRawJCL TracerRawMadExcept TracerLeaksRAW TracerFramesEurekaLogV7 TracerFramesEurekaLog

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 584 when you don't know which class do you need.

Tip
Use GetTracer 331 function instead (when possible).

See also
- GetTracer 331

9.87 GUIFont

Returns font for GUI applications.

Unit
EFonts 85

Syntax

Code (Delphi)
```
function GUIFont: TFont;
```

Return value
Variable-pitch (proportional space) system font.

Remarks
This function returns font from SystemParametersInfo(SPI_GETNONCLIENTMETRICS).lfMessageFont.

See also
- What are SYSTEM_FONT and DEFAULT_GUI_FONT?
- MonoFont 386

9.88 HandleException

Handles exception.

Unit
EBase 52

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
function HandleException(
   const E: TObject;
   const AExpected: Boolean = False;
   const AHandler: String = ''
): Integer;
```

Parameters

E [in]

Exception object to process. Does nothing for EAbort.

Return value

Error code from E. You can use this code to specify failure reason for thread, process, etc. This code is never equals 0.

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 205.

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

- TExceptionManager Class 203
- ExceptionManager Function 305
- ExceptionManager.Handle Method 205

9.89 HookWin32API

Adds a function to resource monitoring.

Unit

EResLeaks 102

Syntax

Code (Delphi)

```
function HookWin32API(
   const AModuleName, AAPIName, AAPICategory, AAPIType: String;
   AParamCount: Integer;
   ACreate: Boolean;
   ACheckType: TCheckType 489 = ctEqual;
   ACheckValue: PtrUInt = 0;
   AResultParam: Integer = -1
): Pointer;
```

Parameters

AModuleName

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

- 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)

9.90 IfElse

A replacement for ternary `if` operator.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function IfElse(
   const AOK: Boolean;
   const AStr1: String;
   const AStr2: String = ''
): String;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AOK
Condition.

AStr1
Result for `True`.

AStr2
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++.

9.91 IniReadString

Reads string from .ini file.

Unit

EConfig 57

Syntax

Code (Delphi)

```
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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 286.

Remarks

Warning

This is low-level routine. Use it only to implement your own custom high-level wrapper.

Use ConfigReadString 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

See also

- IniWriteString 340
- ConfigReadString 276

9.92 InitCheckFreeze

Starts checking for freezes and hangs.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
procedure InitCheckFreeze(
   const AThreadID: Cardinal = 0;
   const AMainWnd: HWND = 0;
   const AThreadHandle: THandle = 0
);
```

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`.

Remarks

This function creates a background freeze detection thread. Thread will use current EurekaLog options.

The function will use FreezeThreadClass 585 global variable to determinate which class should be used as background freeze detection thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 291 to stop background freeze detection thread.

See also

- DoneCheckFreeze Function 291
- PauseCheckFreeze Function
- TFreezeThread Class 217
- FreezeThreadClass Variable 585

9.93 InitSalt

Generates random salt.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function InitSalt: TSalt 554;
```

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 428 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 443 function and TwofishDeriveKey 455 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 := 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);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   // 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 443
- TwofishDeriveKey Function 455

9.94 InitWER

Initializes EWER unit.

Unit

EWER 120

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.

9.95 IniWriteString

Writes string to .ini file.

Unit

EConfig 57

Syntax

Code (Delphi)

```
procedure IniWriteString(
   const AFileName: String;
   const ASection, AName, AValue: String
); deprecated'Consider using ConfigWriteXXX/ConfigReadXXX routines';
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AFileName
A full absolute path to file name. If file doesn't exist - it'll be created (as unicode .ini file).

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 295, 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 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

See also

- IniReadString 336
- ConfigWriteString 281

9.96 InstructionToString

Converts decoded CPU instruction to string represenstation.

Unit

EDisAsm 70

Syntax

Code (Delphi)

```
function InstructionToString(
   const AInstr: TInstruction 537;
   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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

CPU (such as EDisAsmX8632.pas, EDisAsmX8664.pas, etc.).

Return value
Textual representation of `AInstr`.

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 290
- DecodeInstruction function 287
- DisasmFunction function 289

9.97 IntoIDE

Detects if we're running inside IDE.

Unit
ELowLevel 93

Syntax

```
function IntoIDE: 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.

9.98 IsAppType

Checks if application is of specified type.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EAppType 50

Syntax

Code (Delphi)

```
function IsAppType(
   const AAppType: TAppType 482
): 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 429 procedure to indicate that they are installed.

There are also simple wrappers for more easy access. For example, IsVCL 366, IsConsole 347 and so on. See EAppType 50 unit for more info.

See also

- SetAppType 429

9.99 IsBPL

Checks if specified module is BPL file.

Unit

EAppType 50

Syntax

Code (Delphi)

```
function IsBPL(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   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 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 327
- IsDLL 349
- IsModuleLibrary 359

9.100 IsCGI

Checks if application is CGI application.

Unit

EAppType 50

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 342
- IsCGIWin 345
- IsCGIConsole 345

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.101 IsCGIConsole

Checks if application is CGI console application.

Unit

EAppType 50

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.

This is a wrapper for IsAppType 342 function.

Returns "`IsCGI and IsConsole`".

If this function returns True, then IsCGI 344 also returns True. Opposite is not correct.

See also

- IsAppType 342
- IsCGI 344
- IsConsole 347
- IsCGIWin 345

9.102 IsCGIWin

Checks if application is CGI Windows application.

Unit

EAppType 50

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.

© 2001-2015, 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 342 function.

Returns "`IsCGI and (not IsConsole)`".

If this function returns True, then IsCGI 344 also returns True. Opposite is not correct.

See also

- IsAppType 342
- IsCGI 344
- IsConsole 347
- IsCGIConsole 345

9.103 IsCLX

Checks if application is CLX application.

Unit

EAppType 50

Syntax

Code (Delphi)

```
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 342 function.

CLX application is Qt cross-platform application in Delphi/C++ Builder 6 and 7.

See also

- IsAppType 342
- IsVCL 366
- IsFMX 355
- IsCPL 346

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.104 IsCompiledWithPackages

Checks if this module is compiled with packages.

Unit
ELowLevel 93

Syntax

Code (Delphi)

```
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.

9.105 IsConsole

Checks if application is console application.

Unit
EAppType 50

Syntax

Code (Delphi)

```
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 342
- IsCGIConsole 345

9.106 IsCPL

Checks if application is control panel application.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EAppType 50

Syntax

Code (Delphi)
```
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 342 function.

CPL application is a DLL applet for Control Panel with .cpl extension.

See also
- IsAppType 342
- IsVCL 366
- IsFMX 366
- IsCLX 346

9.107 IsDebugged

Determines whether the calling process is being debugged by a user-mode debugger.

Unit
ELowLevel 93

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.108 IsDLL

Checks if specified module is DLL file.

Unit

EAppType 50

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

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 327
- IsBPL 343
- IsModuleLibrary 359

9.109 IsEurekaLogActive

Indicates if EurekaLog is active or disabled in this application.

Unit

EBase 52

Syntax

Code (Delphi)

```
function IsEurekaLogActive: Boolean;
```

Return value

`True` if EurekaLog is active, `False` - otherwise.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

EurekaLog is considered to be active if EurekaLog should do its work. You can control EurekaLog activation by calling SetEurekaLogState 431 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 431 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 354 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');
```

See also

- IsEurekaLogInstalled 354
- IsEurekaLogInstalledInThread
- SetEurekaLogState 431
- SetEurekaLogStateInThread 7
- _IsEurekaLogActive
- _InstallEurekaLog

9.110 IsEurekaLogActive

Indicates if EurekaLog is active or disabled in this application.

Unit

ExceptionLog7 121

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 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 354 function.

Note

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

Note

This is a wrapper for IsEurekaLogActive 349 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- _InstallEurekaLog

9.111 IsEurekaLogActiveInThread

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

Unit

ExceptionLog7 121

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- SetEurekaLogState 6
- SetEurekaLogStateInThread 7
- _InstallEurekaLog
- _IsEurekaLogActive

9.112 IsEurekaLogInstalled

Indicates if EurekaLog is installed in this application.

Unit

ExceptionLog7 121

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 121);
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 354 function.

Examples

Code (Delphi)

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

Code (Delphi)

```
{$IFDEF EUREKALOG}
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

See also

- IsEurekaLogActive 3
- _InstallEurekaLog

9.113 IsEurekaLogInstalled

Indicates if EurekaLog is installed in this application.

Unit

EBase 52

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 121);
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

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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
{$ENDIF}
```

See also

- IsEurekaLogActive 349

9.114 IsEXE

Checks if specified module is EXE file.

Unit

EAppType 50

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 327 function.

See also

- GetModuleType 327

9.115 IsFMX

Checks if application is FMX application.

Unit

EAppType 50

Syntax

Code (Delphi)

```
function IsFMX: Boolean;
```

© 2001-2015, 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 342 function.

FMX is FireMonkey/FM cross-platform application in Delphi/C++ Builder XE2+.

See also

- IsAppType 342
- IsCLX 346
- IsVCL 366
- IsCPL 346

9.116 IsGUI

Checks if application is GUI application.

Unit

EAppType 50

Syntax

Code (Delphi)

```
function IsGUI: Boolean;
```

Return value

Returns True if application is GUI 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 342 function.

GUI application is VCL, CLX, VCLX or CPL application.

Returns "`IsVCL or IsCLX or IsFMX or IsCPL`".

See also

- IsAppType 342
- IsVCL 366
- IsCLX 346
- IsFMX 355

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- IsCPL 347
- IsConsole 347
- IsService 361

9.117 IsIntraWeb

Checks if application is IntraWeb application.

Unit
EAppType 50

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 342

9.118 IsISAPI

Checks if application is ISAPI application.

Unit
EAppType 50

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- IsAppType 342

9.119 IsMemLeaksEnabled

Indicate if memory leaks checks are enabled.

Unit

EMemLeaks 96

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 96.

This is a global state, which can be changed via EnableMemLeaksRegistering 294/ DisableMemLeaksRegistering 288 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 358.

Tip

This function also returns `False` when inside EurekaLog's own code.

See also

- EnableMemLeaksRegistering Function 294
- DisableMemLeaksRegistering Function 288
- SetMemLeaksRegistering Function
- IsMemLeaksInstalled Function 358

9.120 IsMemLeaksInstalled

Indicate if EurekaLog's memory filter is installed.

Unit

EMemLeaks 96

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

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 96.

Note

This value is initialized on startup and never changes.

See also

- IsMemLeaksEnabled Function 358

9.121 IsModuleLibrary

Checks if specified module is DLL or BPL file.

Unit

EAppType 50

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- GetModuleType 327
- IsDLL 349
- IsBPL 343

9.122 IsReadableFile

Verifies that the caller has read access to the specified file.

Unit
ELowLevel 93

Syntax

Code (Delphi)
```
function IsReadableFile(
   const AFileName: String
): Boolean;
```

Parameters
AFileName
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 373

9.123 IsRTLCompiledWithStackFrames

Determines whether the RTL was compiled with Stack Frames option enabled.

Unit
ELowLevel 93

Syntax

Code (Delphi)
```
function IsRTLCompiledWithStackFrames: Boolean;
```

Return value
If RTL was compiled with Stack Frames option, the return value is True.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

If RTL was compiled without Stack Frames option, the return value is `False`.

Remarks

This is a simple wrapper for IsStackFramesProc 361 with arbitrary simple function from RTL.

Warning
This is heuristic function.

See also

- IsStackFramesProc 361

9.124 IsService

Checks if application is a win32 service application.

Unit

EAppType 50

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 342 function.

Service application can contain one or more Win32 services.

See also

- IsAppType 342
- IsVCL 366
- IsConsole 347

9.125 IsStackFramesProc

Determines whether the specified function has a stack frame.

Unit

ELowLevel 93

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
function IsStackFramesProc(
   const AAddr: Pointer
): Boolean;
```

Parameters

AAddr

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 360

9.126 IsValidBlockAddr

Verifies that the caller has read access to the specified range of memory.

Unit

ELowLevel 93

Syntax

```
Code (Delphi)
function IsValidBlockAddr(
   const Address: Pointer;
   const Size: Cardinal
): Boolean;
```

Parameters

Address

A pointer to the first byte of the memory block.

Size

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 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 329 function.

See GetReadableSize 329 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 329
- IsValidCodeAddr 364
- IsValidClass 363
- IsValidObject 365
- IsValidHandle 365
- IsValidRunningThread
- IsValidSymbolName

9.127 IsValidClass

Verifies that the specified pointer is a valid class.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function IsValidClass(
   ClassType: TClass
): Boolean;
```

Parameters

`ClassType`

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Warning

Use this routine with caution. It's heuristic.

See also

- IsValidBlockAddr 362
- IsValidCodeAddr 364
- IsValidObject 365
- IsValidHandle 365
- IsValidRunningThread
- IsValidSymbolName

9.128 IsValidCodeAddr

Verifies that the caller has execute access to the specified range of memory.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function IsValidCodeAddr(
   const AAddress: Pointer;
   const ASize: Cardinal
): Boolean;
```

Parameters

AAddress

A pointer to the first byte of the memory block.

ASize

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 329 function.

See GetReadableSize 329 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- GetReadableSize 329
- IsValidBlockAddr 362
- IsValidClass 363
- IsValidObject 365
- IsValidHandle 365
- IsValidRunningThread
- IsValidSymbolName

9.129 IsValidHandle

Verifies that this is a valid kernel handle.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function IsValidHandle(
   const AHandle: THandle
): Boolean;
```

Parameters

AHandle

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 362
- IsValidCodeAddr 364
- IsValidClass 363
- IsValidObject 365
- IsValidRunningThread
- IsValidSymbolName

9.130 IsValidObject

Verifies that the specified pointer is a valid object.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
function IsValidObject(
   const AObj: TObject
): Boolean;
```

Parameters

AObj

An object pointer.

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 362
- IsValidCodeAddr 364
- IsValidClass 363
- IsValidHandle 365
- IsValidRunningThread
- IsValidSymbolName

9.131 IsVCL

Checks if application is VCL application.

Unit

EAppType 50

Syntax

Code (Delphi)

```
function IsVCL: Boolean;
```

Return value

Returns True if application is VCL application. False - otherwise or on error.

© 2001-2015, 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 342 function.

VCL is default usual application in Delphi/C++ Builder for Win32.

See also

- IsAppType 342
- IsCLX 346
- IsFMX 355
- IsCPL 346

9.132 IsWeb

Checks if application is web application.

Unit

EAppType 50

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 342 function.

Web application is CGI, ISAPI or IntraWeb application.

Returns "`IsCGI or IsISAPI or IsIntraWeb`".

See also

- IsAppType 342
- IsCGI 344
- IsISAPI 357
- IsIntraWeb 357

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.133 IsWin10

Detect Windows 10 (minimum).

Unit
ESysInfo 115

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

This function can be used to determinate minimum required level of operating system.

See also
- IsWin81 369
- IsWin8 369
- IsWin7 369
- IsWinVista 371
- IsWinXP 371
- IsWin64 368
- IsWorkstation 372

9.134 IsWin64

Detect OS bitness.

Unit
ESysInfo 115

Syntax

Code (Delphi)

```
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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Tip
The result of this function doesn't depend on bitness of calling application.

See also
- IsWin10 368
- IsWin81 370
- IsWin8 369
- IsWin7 369
- IsWinVista 371
- IsWinXP 371
- IsWorkstation 372

9.135 IsWin7

Detect Windows 7 (minimum).

Unit
ESysInfo 115

Syntax

Code (Delphi)
```
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.

See also
- IsWin10 368
- IsWin81 370
- IsWin8 369
- IsWinVista 371
- IsWinXP 371
- IsWin64 368
- IsWorkstation 372

9.136 IsWin8

Detect Windows 8 (minimum).

Unit
ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

- IsWin10 368
- IsWin81 370
- IsWin7 369
- IsWinVista 371
- IsWinXP 371
- IsWin64 368
- IsWorkstation 372

9.137 IsWin81

Detect Windows 8.1 (minimum).

Unit

ESysInfo 115

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

This function can be used to determinate minimum required level of operating system.

See also
- IsWin10 368
- IsWin8 369
- IsWin7 369
- IsWinVista 371
- IsWinXP 371
- IsWin64 368
- IsWorkstation 372

9.138 IsWinVista

Detect Windows Vista (minimum).

Unit
ESysInfo 115

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

This function can be used to determinate minimum required level of operating system.

See also
- IsWin10 368
- IsWin81 370
- IsWin8 369
- IsWin7 369
- IsWinXP 371
- IsWin64 368
- IsWorkstation 372

9.139 IsWinXP

Detect Windows XP (minimum).

Unit
ESysInfo 115

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

- IsWin10 368
- IsWin81 370
- IsWin8 369
- IsWin7 369
- IsWinVista 371
- IsWin64 368
- IsWorkstation 372

9.140 IsWorkstation

Gets server/workstation status of current computer.

Unit

ESysInfo 115

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- IsWin10 368
- IsWin81 370
- IsWin8 369
- IsWin7 369
- IsWinVista 371
- IsWinXP 371
- IsWin64 368

9.141 IsWritableFile

Verifies that the caller has write access to the specified file.

Unit
ELowLevel 93

Syntax

Code (Delphi)

```
function IsWritableFile(
   const AFileName: String
): Boolean;
```

Parameters
AFileName
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.

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 360

9.142 LastBugID

Returns BugID of last exception in current thread.

Unit
ExceptionLog7 121

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
function LastBugID: TBugID 484;
```

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

See also
- BugID

9.143 MainThreadTopOfStack

Returns top of the stack for the main thread.

Unit
ELowLevel 93

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.

9.144 MD5Hash

Calculates MD5 hash.

Unit
EHash 87

Syntax

Code (Delphi)

```
function MD5Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TMD5Hash 540; overload;
```

Parameters
AData [in]
A data to calculate hash.

ADataSize [in]
Size of AData in bytes.

Return value
MD5 hash of the specified AData.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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).

9.145 MemLeaksName

Names a memory block.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure MemLeaksName(
   const ABlock: Pointer;
   const AName: String
);
```

Parameters

ABlock [in]

A pointer to memory block.

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 492 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 376 and MemLeaksNameStr 376 for dynamic arrays and strings instead of this function.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- MemLeaksOwn Function 377
- MemLeaksNameDynArray Function 376
- MemLeaksNameStr Function 376
- DumpAllocationsToFile Function 292

9.146 MemLeaksNameDynArray

Names a memory block (variant for dynamic arrays).

Unit

EMemLeaks 96

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 375, but it accepts only pointers to dynamic arrays.

See also

- MemLeaksName Function 375

9.147 MemLeaksNameStr

Names a memory block (variant for strings).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure MemLeaksNameStr(
   const ABlock: Pointer;
   const AName: String
);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 375, but it accepts only pointers to strings.

See also

- MemLeaksName Function 375

9.148 MemLeaksOwn

Sets ownership for memory block.

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure MemLeaksOwn(
   const AParentBlock, AChildBlock: Pointer;
   const AName: String = ''
);
```

Parameters

AParentBlock [in]
A pointer to parent block.

AChildBlock [in]
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 380 and MemLeaksName 375 functions. MemLeaksOwn will name AChildBlock with AName (as MemLeaksName 375 do) and sets AParentBlock as a parent to AChildBlock (as MemLeaksSetParentBlock 380 do).

The difference is that MemLeaksOwn will not overwrite a parent link in AChildBlock if that link was already set. Whereas MemLeaksSetParentBlock 380 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 380 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 378.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

MemLeaksOwnDynArrayParent 379, MemLeaksOwnDynArrayBoth 378 and MemLeaksOwnStr 380 for dynamic arrays and strings instead of this function.

See also

- MemLeaksName Function 375
- MemLeaksSetParentBlock Function 380
- MemLeaksOwnDynArrayChild Function 378
- MemLeaksOwnDynArrayParent Function 379
- MemLeaksOwnDynArrayBoth Function 378
- MemLeaksOwnStr Function 380
- DumpAllocationsToFile Function 292

9.149 MemLeaksOwnDynArrayBoth

Sets ownership for memory block (variant for dynamic arrays).

Unit

EMemLeaks 96

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.

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 377, but it accepts only pointers to dynamic arrays.

See also

- MemLeaksOwn Function 377

9.150 MemLeaksOwnDynArrayChild

Sets ownership for memory block (variant for dynamic arrays).

Unit

EMemLeaks 96

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 377, but it accepts only pointers to dynamic arrays.

See also

- MemLeaksOwn Function 377

9.151 MemLeaksOwnDynArrayParent

Sets ownership for memory block (variant for dynamic arrays).

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure MemLeaksOwnDynArrayParent(
   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 377, but it accepts only pointers to dynamic arrays.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- MemLeaksOwn Function 377

9.152 MemLeaksOwnStr

Sets ownership for memory block (variant for strings).

Unit

EMemLeaks 96

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 377, but it accepts only pointers to strings.

See also

- MemLeaksOwn Function 377

9.153 MemLeaksSetParentBlock

Unit

EMemLeaks 96

Syntax

Code (Delphi)

```
procedure MemLeaksSetParentBlock(
   const AParentBlock, AChildBlock: Pointer
);
```

See also

- MemLeaksOwn Function 377
- MemLeaksSetParentBlockDynArrayChild Function 381
- MemLeaksSetParentBlockDynArrayParent Function 382

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- MemLeaksSetParentBlockDynArrayBoth Function 381
- MemLeaksSetParentBlockStr Function 382
- DumpAllocationsToFile Function 292

9.154 MemLeaksSetParentBlockDynArrayBoth

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

Unit
EMemLeaks 96

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 380, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 380

9.155 MemLeaksSetParentBlockDynArrayChild

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

Unit
EMemLeaks 96

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks
This function is the same as MemLeaksSetParentBlock function 380, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 380

9.156 MemLeaksSetParentBlockDynArrayParent

Sets parent-child relation between two memory blocks for grouping (variant for dynamic arrays).

Unit
EMemLeaks 96

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 380, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 380

9.157 MemLeaksSetParentBlockStr

Sets parent-child relation between two memory blocks for grouping (variant for strings).

Unit
EMemLeaks 96

Syntax

Code (Delphi)
```
procedure MemLeaksSetParentBlockStr(
   const AParentBlock, AChildBlock: Pointer
);
```

Parameters
AParentBlock [in]
A pointer to parent block.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

AChildBlock [in]
A string casted to pointer.

Remarks
This function is the same as MemLeaksSetParentBlock function 380, but it accepts only pointers to strings.

See also
- MemLeaksSetParentBlock Function 380

9.158 MemProtect

Encrypts process memory.

Unit
EEncrypt 73

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 525
); overload;

procedure MemProtect(
   const AData: TEncryptBuffer 525;
   out AProtectedData: TEncryptBuffer 525
); 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 428 function instead).

AProtectedDataSize [out]
Size of AProtectedData.

AData [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 295 or SecureFree 428 functions.

AProtectedData [out]
Encrypted data (size will be different). Delete this data by EncryptBufferFree 295 or SecureFree 428 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 TEAEncrypt 447 or RSAEncrypt 414 functions.

Call the MemUnprotect 384 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.

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 384
- TEAEncrypt Function 447
- RSAEncrypt Function 414

9.159 MemUnprotect

Decrypts process memory.

Unit
EEncrypt 73

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 525
); overload;

procedure MemUnprotect(
   const AProtectedData: TEncryptBuffer 525;
   out AData: TEncryptBuffer 525
); 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 428 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 295 or SecureFree 428 functions.

AData [out]
Unencrypted sensitive data (size will be different). Delete this data by EncryptBufferFree 295 or SecureFree 428 functions.

Return value

Unencrypted sensitive data (size will be different).

Remarks

MemUnprotect function decrypts memory that was encrypted using the MemProtect 383 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 383
- TEADecrypt Function 437
- RSADecrypt Function 411

9.160 MonoFont

Returns fixed-width font.

Unit

EFonts 85

Syntax

Code (Delphi)

```
function MonoFont: TFont;
```

Return value

Fixed-pitch (monospace) system font.

Remarks

This function returns font from GetStockObject(ANSI_FIXED_FONT).

See also

- GUIFont 333

9.161 NameThread

Alias for NameThreadForDebugging to avoid conflicts with System.Classes.TThread.NameThreadForDebugging.

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
procedure NameThread(
   const AName: String;
   const AThreadID: Cardinal = 0
);
```

Parameters

AName [in]
Thread's name.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

AThreadID [in, optional]
TID (Thread ID) of the thread to set new name.

Remarks

See NameThreadForDebugging 387 for a detailed description. `NameThread` is the same function as `NameThreadForDebugging`.

See also

- NameThreadForDebugging Function 387

9.162 NameThreadForDebugging

Sets the name for a thread.

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
procedure NameThreadForDebugging(
   const AName: String;
   const AThreadID: Cardinal = 0
);
```

Parameters

AName [in]
A new name for the thread identified via `AThreadID` argument.

AThreadID [in, optional]
TID (Thread ID) of the thread to set new name for. 0 means current thread.

Remarks

Use `NameThreadForDebugging` to name a thread that was created without a name.

Also, use this function to rename a specified thread. `AName` is the string specifying the new name. `AThreadID` is the TID identifier of the thread to be named. If `AThreadID` is 0 or -1, the current thread is renamed.

A thread name is also used to specify thread name for the debugger's Thread Status tab.

Note

This function is different from the default RTL's System.Classes.TThread.NameThreadForDebugging . This function will update internal EurekaLog thread information storage, so thread name will be used at run-time to insert thread information into bug reports. Therefore this function completes both tasks: sets name for the debugger and stores name for later use in bug reports. Therefore it is recommended to replace existing calls to `System.Classes.TThread.NameThreadForDebugging` with calls to this function.

See also

- NameThread Function 386

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.163 PageSize

Returns current memory's page size.

Unit
ELowLevel 93

Syntax

Code (Delphi)

```
function PageSize: Cardinal;
```

Return value
Page size in bytes.

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
- AllocatePageAlign 260

9.164 PauseFreezeCheck

Temporarily disables freeze checks.

Unit
EFreeze 86

Syntax

Code (Delphi)

```
procedure PauseFreezeCheck;
```

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 410

Tip

PauseFreezeCheck can be called multiple times by the same thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- ResumeFreezeCheck Function 410
- DoneCheckFreeze Function 291
- TFreezeThread.ChecksPaused Property 218

9.165 PrimaryInstance

The same as MainInstance. See remarks for more info.

Unit
ELowLevel 93

Syntax

Code (Delphi)
```
function PrimaryInstance: HMODULE;
```

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 286
- HInstance

9.166 RaiseExpected

Raises expected exception.

Unit
EBase 52

Syntax

Code (Delphi)
```
procedure RaiseExpected(
   const AException: Exception;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID 484 = 0
); overload;

procedure RaiseExpected(
   const AException: Exception;
   const AURL: String;
   const ABugID: TBugID 484 = 0
); overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 170 and BugID 164 properties.

See also

- ShowExcepted

9.167 RaiseFreezeException

Raises freeze/hang/deadlock exception for the specified thread.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
procedure RaiseFreezeException(
   const AThreadID: Cardinal = 0;
   const AThreadHandle: THandle = 0
);
```

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`.

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- InitCheckFreeze Function 337
- TFreezeThread Class 217
- FreezeThreadClass Variable 585

9.168 RegisterDialogClass

Registers dialog class.

Unit

EDialog 64

Syntax

Code (Delphi)

```
procedure RegisterDialogClass(
   const AClass: TBaseDialogClass 482
);
```

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 188.

Note

This function adds dialog class to the end 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

- RegisterDialogClassFirst 391
- TBaseDialog 32
- FindDialogClass 307
- CurrentEurekaLogOptions

9.169 RegisterDialogClassFirst

Registers dialog class.

Unit

EDialog 64

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure RegisterDialogClassFirst(
   const AClass: TBaseDialogClass 482
);
```

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 188.

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 391
- TBaseDialog 32
- FindDialogClass 307
- CurrentEurekaLogOptions

9.170 RegisterEventAttachedFilesRequest

Registers OnAttachedFilesRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventAttachedFilesRequest(
   const AHandler: TELEvAttachedFilesRequestMeth 19;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventAttachedFilesRequest(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const ACustom: Pointer;
   const AHandler: TELEvAttachedFilesRequestProc 495;
   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 75
- Unregistering event handler 461

9.171 RegisterEventBeginReportGen

Registers OnBeginReportGen event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventBeginReportGen(
   const AHandler: TELEvReportGenMeth 26;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventBeginReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 522;
   const AFirst: Boolean = False
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
); 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 75
- Unregistering event handler 462

9.172 RegisterEventCustomBugID

Registers OnCustomBugID event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventCustomBugID(
   const AHandler: TELEvCustomBugIDMeth 22;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomBugID(
   const ACustom: Pointer;
   const AHandler: TELEvCustomBugIDProc 496;
   const AFirst: Boolean = False
); overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 462

9.173 RegisterEventCustomButtonClick

Registers OnCustomButtonClick event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventCustomButtonClick(
   const AHandler: TELEvCustomButtonClickMeth 24;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvCustomButtonClickProc 498;
   const AFirst: Boolean = False
); overload;
```

Parameters

AHandler [in]
Event handler to register. Must be assigned.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 463

9.174 RegisterEventCustomDataRequest

Registers OnCustomDataRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventCustomDataRequest(
   const AHandler: TELEvCustomDataRequestMeth 23;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomDataRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomDataRequestProc 500;
   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).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 464

9.175 RegisterEventCustomFileName

Registers OnCustomFileName event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventCustomFileName(
   const AHandler: TELEvCustomFileNameMeth 21;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomFileName(
   const ACustom: Pointer;
   const AHandler: TELEvCustomFileNameProc 501;
   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*)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 465

9.176 RegisterEventCustomWebFieldsRequest

Registers OnCustomWebFieldRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventCustomWebFieldsRequest(
   const AHandler: TELEvCustomWebFieldsRequestMeth 23;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventCustomWebFieldsRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomWebFieldsRequestProc 503;
   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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 466

9.177 RegisterEventEndReportGen

Registers OnEndReportGen event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventEndReportGen(
   const AHandler: TELEvReportGenMeth 26;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventEndReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 522;
   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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 466

9.178 RegisterEventExceptionAction

Registers OnExceptionAction event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionAction(
   const AHandler: TELEvExceptionActionMeth 15;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionAction(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionActionProc 505;
   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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 467

9.179 RegisterEventExceptionError

Registers OnExceptionError event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionError(
   const AHandler: TELEvExceptionErrorMeth 17;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionError(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionErrorProc 508;
   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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Unregistering event handler 468

9.180 RegisterEventExceptionNotify

Registers OnExceptionNotify event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventExceptionNotify(
   const AHandler: TELEvExceptionNotifyMeth 14;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventExceptionNotify(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionNotifyProc 511;
   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).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- General events concept 75
- Unregistering event handler 469

9.181 RegisterEventHelpButtonClick

Registers OnHelpButtonClick event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventHelpButtonClick(
   const AHandler: TELEvHelpButtonClickMeth 25;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventHelpButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvHelpButtonClickProc 514;
   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 75
- Unregistering event handler 470

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.182 RegisterEventPasswordRequest

Registers OnPasswordRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventPasswordRequest(
   const AHandler: TELEvPasswordRequestMeth 18;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventPasswordRequest(
   const ACustom: Pointer;
   const AHandler: TELEvPasswordRequestProc 516;
   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 75
- Unregistering event handler 470

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.183 RegisterEventRaise

Registers OnRaise event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventRaise(
   const AHandler: TELEvRaiseMeth 12;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventRaise(
   const ACustom: Pointer;
   const AHandler: TELEvRaiseProc 518;
   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 75
- Unregistering event handler 471

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.184 RegisterEventReproduceRequest

Registers OnReproduceRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventReproduceRequest(
   const AHandler: TELEvReproduceRequestMeth 21;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventReproduceRequest(
   const ACustom: Pointer;
   const AHandler: TELEvReproduceRequestProc 523;
   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 75
- Unregistering event handler 472

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.185 RegisterEventZippedFilesRequest

Registers OnZippedFilesRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure RegisterEventZippedFilesRequest(
   const AHandler: TELEvZippedFilesRequestMeth 20;
   const AFirst: Boolean = False
); overload;

procedure RegisterEventZippedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvZippedFilesRequestProc 525;
   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 75
- Unregistering event handler 473

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.186 RegKeyCheck

Checks registry key for existing.

Unit

EConfig 57

Syntax

Code (Delphi)

```
function RegKeyCheck(
   const ARoot: HKey;
   const AKey: String
): Boolean;
```

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.

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 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

See also

- RegKeyRead 408
- RegKeyWrite 409

9.187 RegKeyRead

Reads string from registry key.

Unit

EConfig 57

Syntax

Code (Delphi)

```
function RegKeyRead(
   const ARoot: HKey;
   const AKey, AName: String;
   const ADefault: String = ''
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
): 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).

Returned value does not need to be decoded by DecodeConfigValue 286, 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 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

See also

- RegKeyCheck 408
- RegKeyWrite 409
- ConfigReadString 276

9.188 RegKeyWrite

Writes string to registry key.

Unit

EConfig 57

Syntax

Code (Delphi)

```
procedure RegKeyWrite(
   const ARoot: HKey;
   const AKey, AName, AValue: String
);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 295, because registry can hold exact binary copy of data itself.

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 276 or ConfigWriteString 281 (or their equivalents) to read and write configurations.

See also

- RegKeyCheck 408
- RegKeyRead 408
- ConfigWriteString 281

9.189 ResumeFreezeCheck

Enables freeze checks back.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
procedure ResumeFreezeCheck;
```

Remarks

This function will enable freeze detection previously disabled by a call to PauseFreezeCheck 388.

Note

Freeze detection is not necessary enabled after a call to this function. For example, if

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

`PauseFreezeCheck` was called twice - then freeze detection checks will be enabled only after second call to `ResumeFreezeCheck`.

See also

- PauseFreezeCheck Function 388
- InitCheckFreeze Function 337

9.190 RSADecrypt

Decrypts encrypted data by using RSA cipher.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure RSADecrypt(
   const AKey: TRSAKey 552;
   const AEncryptedData: Pointer;
   out AData: Pointer;
   var ADataSize: Cardinal
); overload;

function RSADecrypt(
   const AKey: TRSAKey 552;
   const AEncryptedData: RawByteString
): RawByteString; overload;

procedure RSADecrypt(
   const AKey: TRSAKey 552;
   var AInPlaceData: TEncryptBuffer 525
); overload;
```

Parameters

AKey [in]
A key for decryption. Comes from RSAGenKey 417 function or RSALoadPrivateKey 418 function.

AEncryptedData [in]
A data to decrypt.

AData [out]
A decrypted `AEncryptedData`'s "clear text" data. Must be deleted with FreeMem function (but we recommend to use SecureFree 428 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.

AInPlaceData [in, out]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

On input: encrypted buffer to decrypt.
On output: "clear text" buffer (same size).

Return value

Decrypted "clear text" data.

Remarks

This function decrypts data previosly encrypted with RSAEncrypt 414 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.

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))
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
          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));

          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 443
- TEAEncrypt Function 447

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.191 RSADestroyKey

Deletes RSA key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure RSADestroyKey(
   var AKey: TRSAKey 552
);
```

Parameters

AKey [in]
RSA key to be released.

Remarks

This function deletes RSA key allocated by RSAGenKey 417 or RSALoadPublicKey 419/RSALoadPrivateKey 418 functions.

Call this function to release RSA key once you have finished working with it.

See also

- RSAGenKey Function 417
- RSALoadPublicKey Function 419
- RSALoadPrivateKey Function 418

9.192 RSAEncrypt

Encrypts "clear text" data by using RSA cipher.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure RSAEncrypt(
   const AKey: TRSAKey 552;
   const AData: Pointer;
   out AEncryptedData: Pointer;
   var ADataSize: Cardinal
); overload;

function RSAEncrypt(
   const AKey: TRSAKey 552;
   const AData: RawByteString
): RawByteString; overload;

procedure RSAEncrypt(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AKey: TRSAKey 552;
   var AInPlaceData: TEncryptBuffer 525
); overload;
```

Parameters

AKey [in]
A key for encryption. Comes from RSAGenKey 417 function or RSALoadPublicKey 419 function.

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 428 function instead). Will have different size from AData.

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).

Return value

Encrypted AData's data in a string form.

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
var
   // (Random) symmetric key to encrypt file
   TEAKey: TTEAKey;
   // Asymmetric key to encrypt symmetric key
   RSAKey: TRSAKey;
   // 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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
        Data := TMemoryStream.Create;
        try

         // Load C:\Text.txt file into Data
         Data.LoadFromFile('C:\Text.txt');

         // 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 411
- RSAGenKey Function 417
- RSALoadPublicKey Function 419

9.193 RSAGenKey

Creates new RSA key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function RSAGenKey(
   const AKeyBitSize: Cardinal = 4096
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
): TRSAKey[552];
```

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.

Return value

Generated RSA key.

Delete this key by using RSADestroyKey[414] function.

Remarks

This function generates a new RSA key (public/private keys pair). The returned key must be deleted by RSADestroyKey[414] function once you have finished working with it.

Resulting key may be used for encryption using RSAEncrypt[414] function or for digital signature using RSASign[422] function.

Resulting key may be exported by using RSASavePublicKey[421] and RSASavePrivateKey[420] functions.

You may import previosly created key instead of generating new key. Use RSALoadPublicKey[419] and/or RSALoadPrivateKey[418] 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.[419]
- RSALoadPrivateKey Function.[418]
- RSADestroyKey Function.[414]
- RSAEncrypt Function.[414]
- RSASign Function.[422]
- RSASavePublicKey Function.[421]
- RSASavePrivateKey Function.[420]

9.194 RSALoadPrivateKey

Imports private key.

Unit

EEncrypt[73]

Syntax

Code (Delphi)

```
function RSALoadPrivateKey(
   const AFileName: String;
   const AType: TRSAExport[551];
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const APassword: String = ''
): TRSAKey 552; overload;

function RSALoadPrivateKey(
   const AStream: TStream;
   const AType: TRSAExport 551;
   const APassword: String = ''
): TRSAKey 552; overload;
```

Parameters

AFileName [in]

A fully qualified absolute file name. File must exist. The file is usually obtained from RSASavePrivateKey 420 function.

AType [in]

AFileName/AStream's type. It must match type specified in RSASavePrivateKey 420 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 414 function.

Remarks

This function loads private key from given file or stream.

See also

- RSASavePrivateKey Function 420
- RSALoadPublicKey Function 419
- RSADestroyKey Function 414
- RSADecrypt Function 411
- RSASign Function 422

9.195 RSALoadPublicKey

Imports public key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function RSALoadPublicKey(
   const AFileName: String;
   const AType: TRSAExport 551
): TRSAKey 552; overload;

function RSALoadPublicKey(
   const AStream: TStream;
   const AType: TRSAExport 551
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
): TRSAKey[552]; overload;
```

Parameters

AFileName [in]
A fully qualified absolute file name. File must exist. The file is usually obtained from RSASavePublicKey[421] function.

AType [in]
AFileName/AStream's type. It must match type specified in RSASavePublicKey[421] 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[414] function.

Remarks

This function loads public key from given file or stream.

See also

- RSASavePublicKey Function[421]
- RSALoadPrivateKey Function[418]
- RSADestroyKey Function[414]
- RSAEncrypt Function[414]
- RSAVerify Function[424]

9.196 RSASavePrivateKey

Exports private key.

Unit

EEncrypt[73]

Syntax

Code (Delphi)

```
procedure RSASavePrivateKey(
   const AKey: TRSAKey[552];
   const AFileName: String;
   const AType: TRSAExport[551];
   const APassword: String = ''
); overload;

procedure RSASavePrivateKey(
   const AKey: TRSAKey[552];
   const AStream: TStream;
   const AType: TRSAExport[551];
   const APassword: String = ''
); overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AKey [in]
RSA key to export. Must come from RSAGenKey 417 or RSALoadPrivateKey 418 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.

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 418 function.

See also

- RSALoadPublicKey Function 419
- RSASavePublicKey Function 421
- RSAGenKey Function 417
- RSADecrypt Function 411
- RSASign Function 422

9.197 RSASavePublicKey

Exports public key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure RSASavePublicKey(
   const AKey: TRSAKey 552;
   const AFileName: String;
   const AType: TRSAExport 551
); overload;

procedure RSASavePublicKey(
   const AKey: TRSAKey 552;
   const AStream: TStream;
   const AType: TRSAExport 551
); overload;
```

Parameters

AKey [in]
RSA key to export. Must come from RSAGenKey 417 or RSALoadPublicKey 419 functions.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

Remarks

This function saves public key from given RSA key to file or stream.

Key can be imported later with RSALoadPublicKey 419 function.

See also

- RSALoadPublicKey Function 419
- RSASavePrivateKey Function 420
- RSAGenKey Function 417
- RSAEncrypt Function 414
- RSAVerify Function 424

9.198 RSASign

Signs data.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function RSASign(
   const AKey: TRSAKey 552;
   const AData: Pointer;
   const ADataSize: Cardinal
): TRSASignature 554; overload;

function RSASign(
   const AKey: TRSAKey 552;
   const AData: TEncryptBuffer 525
): TRSASignature 554; overload;

function RSASign(
   const AKey: TRSAKey 552;
   const AData: RawByteString
): TRSASignature 554; overload;
```

Parameters

AKey [in]
A key for digital signing. Comes from RSAGenKey 417 function or RSALoadPrivateKey 418 function.

AData [in]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Data to be signed.

ADataSize [in]
Size of AData in bytes.

Return value

Digital signature (little-endian). Release it with SecureFree 428 function.

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 424 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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
         SecureFree(Signature);
      end;
   finally
      RSADestroyKey(RSAKey);
   end;
end;
```

See also

- RSAVerify Function 424
- RSAGenKey Function 417
- RSALoadPrivateKey Function 418

9.199 RSAVerify

Verifies digital signature.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function RSAVerify(
   const AKey: TRSAKey 552;
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASignature: Pointer;
   const ASignatureSize: Cardinal
): Boolean; overload;

function RSAVerify(
   const AKey: TRSAKey 552;
   const AData: TEncryptBuffer 525;
   const ASignature: TRSASignature 554
): Boolean; overload;

function RSAVerify(
   const AKey: TRSAKey 552;
   const AData: RawByteString;
   const ASignature: TRSASignature 554
): Boolean; overload;
```

Parameters

AKey [in]
A key for verification. Comes from RSAGenKey 417 function or RSALoadPublicKey 419 function. **Must** match the key used in RSASign 422 function to sign `AData`.

AData [in]
Data to be verified (the same data as the one passed to RSASign 422 function). Must be unmodified (otherwise function will return `False`).

ADataSize [in]
Size of `AData` in bytes.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ASignature [in]
Signature to be verified (the same signature as the one returned by RSASign function). Little-endian.

ASignatureSize [in]
Size of ASignature in bytes.

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 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);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
         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

      finally
         SecureFree(Signature);
      end;
   finally
      RSADestroyKey(RSAKey);
   end;
end;
```

See also

- RSASign Function 422
- RSAGenKey Function 417
- RSALoadPublicKey Function 419

9.200 SafeExec

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

Unit

ELowLevel 93

Syntax

Code (Delphi)

```
procedure SafeExec(
   const AProc: ETypes.TProcedure;
   const ASection: String
);
```

Parameters

AProc

A procedure to execute.

Must no throw exceptions as normal operation. Any exception inside this procedure is considered as fatal.

ASection

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Note

The names of routines (`'Init'` and `'Done'`) are important, because this procedure also changes 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

9.201 SDBMHash

Calculates SDBM hash.

Unit

EHash 87

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
function SDBMHash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSDBMHash555; 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.

9.202 SecureFree

Wipes and releases memory.

Unit

EEncrypt731

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(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   var AList: TStringList
); overload;

procedure SecureFree(
   var IV: TTwofishInitVector 569
); overload;

procedure SecureFree(
   var AKey: TTwofishRAWKey 570
); overload;

procedure SecureFree(
   var AKey: TTEAKey 567
); overload;

procedure SecureFree(
   var AKey: TRSAKey 552
); overload;

procedure SecureFree(
   var ABuffer: TEncryptBuffer 525
); 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`.

9.203 SetAppType

Set a type of this application.

Unit

EAppType 50

Syntax

Code (Delphi)

```
procedure SetAppType(
   const AAppType: TAppType 482
);
```

Parameters

AAppType

Type of application to set. This is one of `EType s.at`*XXX* 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 342 function to check application for being of a known type.

There are also simple wrappers for more easy access. For example, IsVCL 366, IsConsole 347 and so on. See EAppType 50 unit for more info.

See also

- IsAppType 342

9.204 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
procedure SetEurekaLogState(
   const AActivate: 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 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 431 function.

Examples

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

9.205 SetEurekaLogState

Enables or disables EurekaLog in this application.

Unit

EBase 52

Syntax

Code (Delphi)

```
procedure SetEurekaLogState(
   const AActivate: 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 349 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.

Examples

Code (Delphi)

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

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

See also

- IsEurekaLogActive 349
- _IsEurekaLogActive
- SetEurekaLogStateInThread 7

9.206 SetEurekaLogStateInThread

Enables or disables EurekaLog in the specified thread.

Unit

ExceptionLog7 121

Syntax

Code (Delphi)

```
procedure SetEurekaLogStateInThread(
   const AThreadID: Cardinal;
   const AActivate: 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.

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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

9.207 SetupConfig

Changes EurekaLog configuration storage.

Unit

EConfig 57

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 57 unit description for basic desciption of EurekaLog configuration.

Examples

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);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- EConfig 57
- ConfigReadString 276
- ConfigWriteString 281
- ConfigIsRegistry 271

9.208 SHA1Hash

Calculates SHA-1 hash.

Unit

EHash 87

Syntax

Code (Delphi)

```
function SHA1Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSHA1Hash 560; 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-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.

9.209 SHA256Hash

Calculates SHA-2/SHA-256 hash.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EHash 87

Syntax

Code (Delphi)

```
function SHA256Hash(
   const AData: Pointer;
   const ADataSize: Cardinal
): TSHA256Hash 560; 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.

9.210 ShowExpected

Shows expected exception.

Unit
EBase 52

Syntax

Code (Delphi)

```
procedure ShowExpected(
   const AException: Exception;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID 484 = 0
); overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
procedure ShowExpected(
   const AException: Exception;
   const AURL: String;
   const ABugID: TBugID[484] = 0
); overload;

procedure ShowExpected(
   const AMessage: String;
   const AContextNumber: Integer = -1;
   const ABugID: TBugID[484] = 0
); overload;

procedure ShowExpected(
   const AMessage: String;
   const AURL: String;
   const ABugID: TBugID[484] = 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.

9.211 SignatureGetInfo

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

Unit

ESignature[112]

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
function SignatureGetInfo(
   const AFileName: String
): TSignatureInfo 561;
```

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.

Important!

Currently, SignatureGetInfo is unable to retrieve info about timestamps in szOID_TIMESTAMP_TOKEN/szOID_RFC3161_counterSign (which is '1.2.840.113549.1.9.16.1.4') format. This type of timestamp is used by Microsoft to sign its own executables.

SignatureGetInfo supports only szOID_RSA_signingTime ('1.2.840.113549.1.9.5'). This type of signature is commonly used.

Therefore, while SignatureGetInfo will extract common information for any executable, but timestamp information will be available only for executables with szOID_RSA_signingTime attribute.

See also

- TSignatureInfo Record 561

9.212 TEADecrypt

Decrypts encrypted data by using TEA cipher.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure TEADecrypt(
   const AKey: TTEAKey 567;
   const AEncryptedData, AData: Pointer;
   const ADataSize: Cardinal
); overload;

function TEADecrypt(
   const AKey: TTEAKey 567;
   const AEncryptedData: RawByteString
): RawByteString; overload;

procedure TEADecrypt(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AKey: TTEAKey 567;
   AInPlaceData: TEncryptBuffer 525
); overload;
```

Parameters

AKey [in]
A key for decryption from TEADeriveKey 443 function.

AEncryptedData [in]
A data to decrypt. Must not contain salt (see TEADeriveKey 443).

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 443).

AInPlaceData [in, out]
On input: encrypted buffer to decrypt. Must not contain salt (see TEADeriveKey 443).
On output: "clear text" buffer (same size).

Return value

Decrypted "clear text" data.

Remarks

This function decrypts data previosly encrypted with TEAEncrypt 447 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 443 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!

© 2001-2015, 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;
   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;
```

© 2001-2015, 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;
   EncryptedData: TEncryptBuffer;
   FS: TFileStream;
begin
   // Mark buffer as emtpy
   EncryptedData.pbData := nil;
   EncryptedData.cbData := 0;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   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
         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 443
- TEAEncrypt Function 447

9.213 TEADefaultKey

Returns pre-initialized default key for TEA.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function TEADefaultKey: TTEAKey 567;
```

Return value

Key derived from empty password.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This function returns the same value as `TEADeriveKey('', nil)`.

Unlike TEADeriveKey 443 function, `TEADefaultKey` does not perform any calculations and simply uses pre-initialized value. Thus, you should use this function to improve performance.

Note
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 447) 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
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   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 443

9.214 TEADeriveKey

Initializes TEA key from password.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function TEADeriveKey(
   const APassword: String;
   const ASalt: PSalt 481 = nil
): TTEAKey 567; overload;

function TEADeriveKey(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASalt: PSalt 481 = nil
): TTEAKey 567; 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 338 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 447 function once you have finished working with key - to erase it with zeros.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 to derive key.

You do not need to call this function if you already have RAW binary TEA key (TTEAKey 567).

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 447 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.

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 441 function in this case.

Important!

Empty salt (zeroed) is not the same as no salt. Use InitSalt 338 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;
```

© 2001-2015, 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
```

© 2001-2015, 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;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- TEADefaultKey Function 441
- TEAInitSessionKey Function 451
- InitSalt Function 338
- TEAEncrypt Function 447
- TEADecrypt Function 437
- TEADestroyKey Function 447

9.215 TEADestroyKey

Fills TEA key with zeros.

Unit
EEncrypt 73

Syntax

Code (Delphi)
```
procedure TEADestroyKey(
   var AKey: TTEAKey 567
);
```

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 441) is always stored in memory - there is not much meaning to wipe default key after using it.

See also
- TEADeriveKey Function 443

9.216 TEAEncrypt

Encrypts "clear text" data by using TEA cipher.

Unit
EEncrypt 73

Syntax

Code (Delphi)
```
procedure TEAEncrypt(
   const AKey: TTEAKey 567;
   const AData, AEncryptedData: Pointer;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const ADataSize: Cardinal
); overload;

function TEAEncrypt(
   const AKey: TTEAKey 567;
   const AData: RawByteString
): RawByteString; overload;

procedure TEAEncrypt(
   const AKey: TTEAKey 567;
   AInPlaceData: TEncryptBuffer 525
); overload;
```

Parameters

AKey [in]
A key for encryption from TEADeriveKey 443 function.

AData [in]
A data to encrypt.

AEncryptedData [out]
An encrypted AData's data. Must have the same size as AData.

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).

Return value

Encrypted AData's data in a string form.

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 443 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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!

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);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
      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

   // 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)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
var
   Key: TTEAKey;
   Data: TEncryptBuffer;
   FS: TFileStream;
begin
   // Mark buffer as emtpy
   Data.pbData := nil;
   Data.cbData := 0;
   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 443
- TEADecrypt Function 437

9.217 TEAInitSessionKey

Generates random TEA key.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
function TEAInitSessionKey: TTEAKey 567;
```

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 447 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 414 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 443 function to derive symmetric TEA key from string password or any other arbitrary "seed" data.

See also
- RSAEncrypt Function 414

9.218 ThreadData

Returns thread information (current thread only).

Unit
EThreadsManager 117

Syntax

Code (Delphi)

```
function ThreadData: TEurekaThreadData 190;
```

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

There is no need to release the returned object.

There is no need to use locking - because informational object 190 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 AquireThreadData 264 or ThreadDataEnum 453 functions to get information about arbitrary thread.

See also

- AquireThreadData Function 264
- ThreadDataEnum Function 453
- TEurekaThreadData Class 190

9.219 ThreadDataEnum

Enumerates running threads.

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
function ThreadDataEnum(
   const AForceAllActive: Boolean = False
): TEurekaThreadDataList 200;
```

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 200 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 452 function to modify thread's dynamic properties.

See also

- ThreadData Function 452
- AquireThreadData Function 264
- TEurekaThreadData Class 190

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.220 TwofishDecrypt

Decrypts encrypted data by using Twofish cipher.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
procedure TwofishDecrypt(
   const AKey: TTwofishKey 569;
   const AEncryptedData: Pointer;
   out AData: Pointer;
   var ADataSize: Cardinal;
   const AInitVector: PTwofishInitVector 481 = nil
); overload;

function TwofishDecrypt(
   const AKey: TTwofishKey 569;
   const AEncryptedData: RawByteString;
   const AInitVector: PTwofishInitVector 481 = nil
): RawByteString; overload;
```

Parameters

AKey [in]
A key for decryption. Comes from TwofishDeriveKey 455 function or TwofishInitKey 459 function.

AEncryptedData [in]
A data to decrypt.

AData [out]
A decrypted AEncryptedData's "clear text" data. Must be deleted with FreeMem function (but we recommend to use SecureFree 428 function instead). Will have different size from AData.

ADataSize [in]
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 457 function during encryption.

AEncryptedData [in]
Encrypted data (in string form) to decrypt. Must not contain salt (see TwofishDeriveKey 455).

Return value

Decrypted "clear text" data.

Remarks

This function decrypts data previosly encrypted with TwofishEncrypt 457 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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).

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 455
- TwofishInitKey Function 459
- TwofishEncrypt Function 457

9.221 TwofishDeriveKey

Initializes Twofish key from password.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
function TwofishDeriveKey(
   const APassword: String;
   const ASalt: PSalt 481 = nil
): TTwofishKey 569; overload;

function TwofishDeriveKey(
   const AData: Pointer;
   const ADataSize: Cardinal;
   const ASalt: PSalt 481 = nil
): TTwofishKey 569; 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 338 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ADataSize [in]
Size of AData in bytes.

Return value

Precalculated 256-bit Twofish key derived from APassword and (optionally) ASalt.

Delete this key by using TwofishDestroyKey 457 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 459 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 SHA-1 to derive 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 459 to precalculate key from RAW key.

Delete the returned value by using TwofishDestroyKey 457 function when you have finished working with key.

Overloaded variant lets you initialize key from any binary password.

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 459
- InitSalt Function 338
- TwofishEncrypt Function 457
- TwofishDecrypt Function 454
- TwofishDestroyKey Function 457

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.222 TwofishDestroyKey

Deletes Twofish key.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
procedure TwofishDestroyKey(
   var AKey: TTwofishKey 569
);
```

Parameters
AKey [in]
Twofish key to be released.

Remarks
This function deletes Twofish key allocated by TwofishDeriveKey 455 or TwofishInitKey 459 functions.

Call this function to release Twofish key once you have finished working with it.

See also
- TwofishDeriveKey Function 455
- TwofishInitKey Function 459

9.223 TwofishEncrypt

Encrypts "clear text" data by using Twofish cipher.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
procedure TwofishEncrypt(
   const AKey: TTwofishKey 569;
   const AData: Pointer;
   out AEncryptedData: Pointer;
   var ADataSize: Cardinal;
   const AInitVector: PTwofishInitVector 481 = nil
); overload;

function TwofishEncrypt(
   const AKey: TTwofishKey 569;
   const AData: RawByteString;
   const AInitVector: PTwofishInitVector 481 = nil
): RawByteString; overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AKey [in]
A key for encryption. Comes from TwofishDeriveKey 455 function or TwofishInitKey 459 function.

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 428 function instead). Will have different size from AData.

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 TwofishInitIV 459 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.

Return value

Encrypted AData's data in a string form.

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.

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 455
- TwofishInitKey Function 459
- TwofishInitIV Function 459
- TwofishDecrypt Function 454

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.224 TwofishInitIV

Creates new IV.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
function TwofishInitIV: TTwofishInitVector 569;
```

Return value
Random initialization vector.

Remarks
This function creates new random initialization vector (IV). Pass the returned value to TwofishEncrypt 457 function to use CBC mode.

Important!

Do not pass the returned value to TwofishDecrypt 454 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.

See also
- TwofishEncrypt Function 457
- TwofishDecrypt Function 454

9.225 TwofishInitKey

"Imports" (precalculates) Twofish key from RAW key value.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
function TwofishInitKey(
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AKey: TTwofishRAWKey 570
): TTwofishKey 569;
```

Parameters

AKey [in]
256-bit RAW Twofish key.

Return value

Twofish key precalculated from AKey.

Delete this key by using TwofishDestroyKey 457 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 455 function to derive key from password.

Delete the returned value by using TwofishDestroyKey 457 function when you have finished working with key.

See also

- TwofishDeriveKey Function 455
- TwofishEncrypt Function 457
- TwofishDecrypt Function 454
- TwofishDestroyKey Function 457

9.226 TZipCompressionToString

Converts ZIP compression method value to string representation.

Unit

EZip 124

Syntax

Code (Delphi)

```
function TZipCompressionToString(
   Compression: TZipCompression 570
): string;
```

Parameters

Compression
Compression method to get its textual representation.

Return value

String representation of Compression argument.

Remarks

Example return values are:

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- 'Unknown'
- 'Stored'
- 'Shrunk'
- 'Reduced1'
- 'Reduced2'
- 'Imploded'
- 'Tokenized'
- Etc.

9.227 UnRegisterEventAttachedFilesRequest

Unregisters previously registered OnAttachedFilesRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventAttachedFilesRequest(
   const AHandler: TELEvAttachedFilesRequestMeth 19
); overload;

procedure UnRegisterEventAttachedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvAttachedFilesRequestProc 495
); 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 75
- Registering event handler 392

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

9.228 UnRegisterEventBeginReportGen

Unregisters previously registered OnBeginReportGen event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventBeginReportGen(
   const AHandler: TELEvReportGenMeth 26
); overload;

procedure UnRegisterEventBeginReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 522
); 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 75
- Registering event handler 393

9.229 UnRegisterEventCustomBugID

Unregisters previously registered OnCustomBugID event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
procedure UnRegisterEventCustomBugID(
   const AHandler: TELEvCustomBugIDMeth 22
); overload;

procedure UnRegisterEventCustomBugID(
   const ACustom: Pointer;
   const AHandler: TELEvCustomBugIDProc 496
); 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 75
- Registering event handler 394

9.230 UnRegisterEventCustomButtonClick

Unregisters previously registered OnCustomButtonClick event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomButtonClick(
   const AHandler: TELEvCustomButtonClickMeth 24
); overload;

procedure UnRegisterEventCustomButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvCustomButtonClickProc 498
); overload;
```

© 2001-2015, 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 75
- Registering event handler 395

9.231 UnRegisterEventCustomDataRequest

Unregisters previously registered OnCustomDataRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomDataRequest(
   const AHandler: TELEvCustomDataRequestMeth 23
); overload;

procedure UnRegisterEventCustomDataRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomDataRequestProc 500
); 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.

© 2001-2015, 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 75
- Registering event handler 396

9.232 UnRegisterEventCustomFileName

Unregisters previously registered OnCustomFileName event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomFileName(
   const AHandler: TELEvCustomFileNameMeth 21
); overload;

procedure UnRegisterEventCustomFileName(
   const ACustom: Pointer;
   const AHandler: TELEvCustomFileNameProc 501
); 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).

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- General events concept 75
- Registering event handler 397

9.233 UnRegisterEventCustomWebFieldsRequest

Unregisters previously registered OnCustomWebFieldRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventCustomWebFieldsRequest(
   const AHandler: TELEvCustomWebFieldsRequestMeth 23
); overload;

procedure UnRegisterEventCustomWebFieldsRequest(
   const ACustom: Pointer;
   const AHandler: TELEvCustomWebFieldsRequestProc 503
); 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 75
- Registering event handler 398

9.234 UnRegisterEventEndReportGen

Unregisters previously registered OnEndReportGen event handler.

Unit

EEvents 75

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
procedure UnRegisterEventEndReportGen(
   const AHandler: TELEvReportGenMeth 26
); overload;

procedure UnRegisterEventEndReportGen(
   const ACustom: Pointer;
   const AHandler: TELEvReportGenProc 522
); 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 75
- Registering event handler 399

9.235 UnRegisterEventExceptionAction

Unregisters previously registered OnExceptionAction event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventExceptionAction(
   const AHandler: TELEvExceptionActionMeth 15
); overload;

procedure UnRegisterEventExceptionAction(
   const ACustom: Pointer;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
   const AHandler: TELEvExceptionActionProc 505
); 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 75
- Registering event handler 400

9.236 UnRegisterEventExceptionError

Unregisters previously registered OnExceptionError event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventExceptionError(
   const AHandler: TELEvExceptionErrorMeth 17
); overload;

procedure UnRegisterEventExceptionError(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionErrorProc 508
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

ACustom [in, optional]
(*for procedural event handlers only*)

© 2001-2015, 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 75
- Registering event handler 401

9.237 UnRegisterEventExceptionNotify

Unregisters previously registered OnExceptionNotify event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventExceptionNotify(
   const AHandler: TELEvExceptionNotifyMeth 14
); overload;

procedure UnRegisterEventExceptionNotify(
   const ACustom: Pointer;
   const AHandler: TELEvExceptionNotifyProc 511
); 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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

(for procedural event handlers).

See also

- General events concept 75
- Registering event handler 402

9.238 UnRegisterEventHelpButtonClick

Unregisters previously registered OnHelpButtonClick event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventHelpButtonClick(
   const AHandler: TELEvHelpButtonClickMeth 25
); overload;

procedure UnRegisterEventHelpButtonClick(
   const ACustom: Pointer;
   const AHandler: TELEvHelpButtonClickProc 514
); 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 75
- Registering event handler 403

9.239 UnRegisterEventPasswordRequest

Unregisters previously registered OnPasswordRequest event handler.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventPasswordRequest(
   const AHandler: TELEvPasswordRequestMeth 18
); overload;

procedure UnRegisterEventPasswordRequest(
   const ACustom: Pointer;
   const AHandler: TELEvPasswordRequestProc 516
); 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 75
- Registering event handler 404

9.240 UnRegisterEventRaise

Unregisters previously registered OnRaise event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventRaise(
   const AHandler: TELEvRaiseMeth 12
); overload;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
procedure UnRegisterEventRaise(
   const ACustom: Pointer;
   const AHandler: TELEvRaiseProc 518
); 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 75
- Registering event handler 405

9.241 UnRegisterEventReproduceRequest

Unregisters previously registered OnReproduceRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventReproduceRequest(
   const AHandler: TELEvReproduceRequestMeth 21
); overload;

procedure UnRegisterEventReproduceRequest(
   const ACustom: Pointer;
   const AHandler: TELEvReproduceRequestProc 523
); overload;
```

Parameters

AHandler [in]
Event handler to unregister.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Registering event handler 406

9.242 UnRegisterEventZippedFilesRequest

Unregisters previously registered OnZippedFilesRequest event handler.

Unit

EEvents 75

Syntax

Code (Delphi)

```
procedure UnRegisterEventZippedFilesRequest(
   const AHandler: TELEvZippedFilesRequestMeth 20
); overload;

procedure UnRegisterEventZippedFilesRequest(
   const ACustom: Pointer;
   const AHandler: TELEvZippedFilesRequestProc 525
); 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75
- Registering event handler 407

9.243 WCTDone

Finalize EWCT unit.

Unit

EWCT 119

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.

9.244 WCTInit

Initialize EWCT unit.

Unit

EWCT 119

Syntax

Code (Delphi)

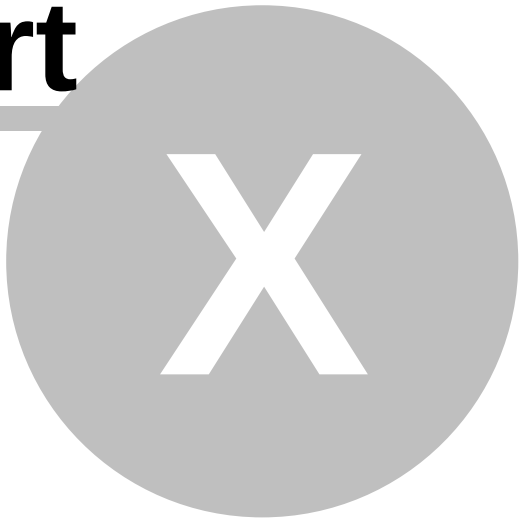
```
procedure WCTInit;
```

Remarks

This function imports all dynamically linked routines. Call this function before using EWCT unit.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



10 All Types

PCustomName 480	Defines buffer for storing name of memory block.
PInstruction 480	A pointer to TInstruction 537
PPointerRecord 480	A pointer to TPointerRecord type 545
PSalt 481	Pointer to salt.
PTwofishInitVector 481	Pointer to IV for Twofish.
TAppType 482	Defines application type.
TBaseDialogClass 482	Pointer to dialog class.
TBlockType 483	Defines type of memory block.
TBugID 484	BugID type.
TCertificateInfo 485	Certificate information.
TCheckType 489	Function check kind.
TCRC16Hash 491	CRC-16 checksum.
TCRC32Hash 492	CRC-32 checksum.
TCustomName 492	Defines buffer for storing name of memory block.
TDumpCompareFunc 493	Comparator for memory blocks.
TDumpFilterFunc 493	Filter for memory blocks.
TELEvAttachedFilesRequestMeth 19	Method event handler for OnAttachedFilesRequest event.
TELEvAttachedFilesRequestProc 495	Procedural event handler for OnAttachedFilesRequest event.
TELEvCustomBugIDMeth 22	Method event handler for OnCustomBugID event.
TELEvCustomBugIDProc 496	Procedural event handler for OnCustomBugID event.
TELEvCustomButtonClickMeth 24	Method event handler for OnCustomButtonClick event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TELEvCustomButtonClickProc 498	Procedural event handler for OnCustomButtonClick event.
TELEvCustomDataRequestMeth 23	Method event handler for OnCustomDataRequest event.
TELEvCustomDataRequestProc 500	Procedural event handler for OnCustomDataRequest event.
TELEvCustomFileNameMeth 21	Method event handler for OnCustomFileName event.
TELEvCustomFileNameProc 501	Procedural event handler for OnCustomFileName event.
TELEvCustomWebFieldsRequestMeth 23	Method event handler for OnCustomWebFieldsRequest event.
TELEvCustomWebFieldsRequestProc 503	Procedural event handler for OnCustomWebFieldsRequest event.
TELEvExceptionActionMeth 15	Method event handler for OnExceptionAction event.
TELEvExceptionActionProc 505	Procedural event handler for OnExceptionAction event.
TELEvExceptionErrorMeth 17	Method event handler for OnExceptionError event.
TELEvExceptionErrorProc 508	Procedural event handler for OnExceptionError event.
TELEvExceptionNotifyMeth 14	Method event handler for OnExceptionNotify event.
TELEvExceptionNotifyProc 511	Procedural event handler for OnExceptionNotify event.
TELEvHelpButtonClickMeth 25	Method event handler for OnHelpButtonClick event.
TELEvHelpButtonClickProc 514	Procedural event handler for OnHelpButtonClick event.
TELEvPasswordRequestMeth 18	Method event handler for OnPasswordRequest event.
TELEvPasswordRequestProc 516	Procedural event handler for OnPasswordRequest event.
TELEvRaiseMeth 12	Method event handler for OnRaise

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EurekaLog CATCH EVERY BUG - EVERY TIME

	event.
TELEvRaiseProc 518	Procedural event handler for OnRaise event.
TELEvReportGenMeth 26	Method event handler for OnReportBeginGen/ OnReportEndGen events.
TELEvReportGenMeth 521	Method event handler for OnReportBeginGen/OnReportEndGen events.
TELEvReportGenProc 521	Procedural event handler for OnReportBeginGen/OnReportEndGen event.
TELEvReportGenProc 522	Procedural event handler for OnReportBeginGen/ OnReportEndGen event.
TELEvReproduceRequestMeth 21	Method event handler for OnReproduceRequest event.
TELEvReproduceRequestProc 523	Procedural event handler for OnReproduceRequest event.
TELEvZippedFilesRequestMeth 20	Method event handler for OnZippedFilesRequest event.
TELEvZippedFilesRequestProc 525	Procedural event handler for OnZippedFilesRequest event.
TEncryptBuffer 525	Contains an arbitrary array of bytes.
TEurekaActionType 526	Action kind for exception processing.
TEurekaCallStack 530	Alias for TEurekaStackListV7.
TEurekaCallStackClass 531	Class of TEurekaBaseStackList.
TEurekaLogFileType 531	Type of file name.
TEurekaModuleType 533	Alias for TModuleType 543
TEurekaStackList 534	Alias for TEurekaStackListV7.
TExceptionNotifyMode 534	Specifies work mode of exception manager.
TFreezeThreadClass 536	Class of freeze detection thread.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TInstruction 537	A CPU instruction.
TMD5Hash 540	MD5 hash.
TMemoryError 540	Type of detected memory error.
TMemoryErrors 543	Defines set of possible memory errors.
TModuleType 543	Indicates kind of executable module.
TPointerRecord 545	Header for memory blocks allocated by EurekaLog.
TResponse 547	No description available
TRSAExport 551	Defines how to export/import RSA key.
TRSAKey 552	Key for RSA encryption.
TRSASignature 554	Digital signature.
TSalt 554	Salt value for deriving key from password.
TSDBMHash 555	SDBM hash.
TSendResult 555	Result kind for exception processing.
TSHA1Hash 560	SHA-1 hash.
TSHA256Hash 560	SHA-2/SHA-256 hash.
TSignatureInfo 561	Information about digital signature of executable file.
TStreamConstructor 567	Function to Create a Compression/ Decompression stream
TTEAKey 567	Key for TEA encryption.
TThreadKind 568	Indicates thread kind.
TTwofishInitVector 569	Initialization Vector (IV) for Twofish.
TTwofishKey 569	(Precalculated) Key for Twofish encryption.
TTwofishRAWKey 570	(RAW) Key for Twofish encryption.
TZipCompression 570	Zip compression method.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TZipEndOfCentralHeader 576	Final block written to zip file
TZipHeader 577	TZipHeader contains information about a file in a zip archive.

10.1 PCustomName

Defines buffer for storing name of memory block.

Unit
EMemLeaks 96

Syntax

Code (Delphi)

```
PCustomName = ^TCustomName 492;
```

Remarks
This is a pointer to TCustomName 492 type.

See also
- TCustomName Type 492
- MemLeaksName Function 375
- IsMemLeaksEnabled Function 358

10.2 PInstruction

A pointer to TInstruction 537.

Unit
EDisAsm 70

Syntax

Code (Delphi)

```
PInstruction = ^TInstruction 537;
```

Remarks
This is a pointer to platform-specific TInstruction 537 record.

See also
- TInstruction Type 537

10.3 PPointerRecord

A pointer to TPointerRecord type 545.

Unit
EMemLeaks 96

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
PPointerRecord = ^TPointerRecord[545];
```

Remarks

This is a pointer to TPointerRecord record[545].

Warning

This data type is subject to change. Consider it to be undocumented.

See also

- TPointerRecord Type[545]
- DumpAllocationsToFile Function[292]
- TDumpCompareFunc Type[493]
- TDumpFilterFunc Type[493]

10.4 PSalt

Pointer to salt.

Unit

EEncrypt[73]

Syntax

```
Code (Delphi)
PSalt = ^TSalt[554];
```

Remarks

Use InitSalt[338] to obtain new random salt. Pass the obtained value to TEADeriveKey[443] function or TwofishDeriveKey[455] 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[443]
- TwofishDeriveKey Function[455]
- InitSalt Function[338]

10.5 PTwofishInitVector

Pointer to IV for Twofish.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
PTwofishInitVector = ^TTwofishInitVector 569;
```

Remarks

Use TwofishInitIV 459 to obtain new random initialization vector. Pass the obtained value to TwofishEncrypt 457 and store it with encrypted data - to be passed into TwofishDecrypt 454 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 457
- TwofishDecrypt Function 454
- TwofishInitIV Function 459

10.6 TAppType

Defines application type.

Unit

ETypes 118

Syntax

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 342
- SetAppType Function 429

10.7 TBaseDialogClass

Pointer to dialog class.

Unit

EDialog 64

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

```
Code (Delphi)
TBaseDialogClass = class of TBaseDialog;
```

Remarks

This type is used to register dialog classes into global list.

See also

- RegisterDialogClass 391
- RegisterDialogClassFirst 391
- FindDialogClass 307

10.8 TBlockType

Defines type of memory block.

Unit

EMemLeaks 96

Syntax

```
Code (Delphi)
TBlockType = (
    btData 484,
    btObject 484,
    btString 484,
    btDynArray 484
);
```

Members

btData 484	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 484	The block is a dynamic array.
btObject 484	The block is an instance of TObject or descendant class.
btString 484	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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- TMemLeaksError Type
- TMemLeaksAdd Type
- TMemLeaksLogBlockProc Type

10.8.1 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

EMemLeaks 96

See also

- TBlockType Class 483

10.8.2 btDynArray

The block is a dynamic array.

Unit

EMemLeaks 96

See also

- TBlockType Class 483

10.8.3 btObject

The block is an instance of TObject or descendant class.

Unit

EMemLeaks 96

See also

- TBlockType Class 483

10.8.4 btString

The block is a string. It can be ANSI or unicode string.

Unit

EMemLeaks 96

See also

- TBlockType Class 483

10.9 TBugID

BugID type.

Unit

ETypes 118

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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

10.10 TCertificateInfo

Certificate information.

Unit

ESignature 112

Syntax

```
TCertificateInfo = record
    Version 489: DWORD;
    Serial 487: String;
    Subject 488: String;
    SubjectID 488: String;
    Issuer 485: String;
    IssuerID 486: String;
    NotBefore 487: TDateTime;
    NotAfter 486: TDateTime;
end;
```

Remarks

This record contains information about digital certificate.

See also

- TSignatureInfo Record 561
- SignatureGetInfo Function 436

10.10.1 Issuer

Issuer of certificate.

Unit

ESignature 112

Syntax

```
Issuer: String;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 486
- Subject Field 488
- TCertificateInfo Class 485

10.10.2 IssuerID

Unique issuer ID.

Unit
ESignature 112

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 485
- SubjectID Field 488
- TCertificateInfo Class 485

10.10.3 NotAfter

Ending date for certificate.

Unit
ESignature 112

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 Class 485

10.10.4 NotBefore

Starting date for certificate.

Unit
ESignature 112

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 Class 485

10.10.5 Serial

Serial number of certificate.

Unit
ESignature 112

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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'
```

See also
- TCertificateInfo Class 485

10.10.6 Subject

Subject name.

Unit
ESignature 112

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 488
- Issuer Field 485
- TCertificateInfo Class 485

10.10.7 SubjectID

Unique subject ID.

Unit
ESignature 112

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

Examples
```
'3DDA323E552DB1FDE5F4FBEE75D6D5B2B187EEDC'
```

See also
- Subject Field 488
- IssuerID Field 486
- TCertificateInfo Class 485

10.10.8 Version

Version of certificate.

Unit
ESignature 112

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 Class 485

10.11 TCheckType

Function check kind.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EResLeaks 102

Syntax

Code (Delphi)

```
TCheckType = (
    ctNone 491,
    ctEqual 490,
    ctNotEqual 491,
    ctHResult 491,
    ctNTStatus 491,
    ctGetLastError 491
);
```

Members

ctEqual 490	Function fails if return value equals to `CheckValue`
ctGetLastError 491	Function fails if return value equals to `CheckValue` **and** `GetLastError` returns value other than `ERROR_SUCCESS`
ctHResult 491	Function returns standard `HResult` value
ctNone 491	Function always successfull
ctNotEqual 491	Function fails if return value wasn't equal to `CheckValue`
ctNTStatus 491	Function returns standard `NTSTATUS` value

Remarks

This type specifies how to determinate if function call was successfull or not. This type is used by HookWin32API 334 function.

10.11.1 ctEqual

Function fails if return value equals to `CheckValue`

Unit

EResLeaks 102

See also

- TCheckType Class 489

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.11.2 ctGetLastError

Function fails if return value equals to `CheckValue` **and** `GetLastError` returns value other than `ERROR_SUCCESS`

Unit

EResLeaks 102

See also

- TCheckType Class 489

10.11.3 ctHResult

Function returns standard `HResult` value

Unit

EResLeaks 102

See also

- TCheckType Class 489

10.11.4 ctNone

Function always successfull

Unit

EResLeaks 102

See also

- TCheckType Class 489

10.11.5 ctNotEqual

Function fails if return value wasn't equal to `CheckValue`

Unit

EResLeaks 102

See also

- TCheckType Class 489

10.11.6 ctNTStatus

Function returns standard `NTSTATUS` value

Unit

EResLeaks 102

See also

- TCheckType Class 489

10.12 TCRC16Hash

CRC-16 checksum.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EHash 87

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 283

10.13 TCRC32Hash

CRC-32 checksum.

Unit
EHash 87

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 284

10.14 TCustomName

Defines buffer for storing name of memory block.

Unit
EMemLeaks 96

Syntax

```
Code (Delphi)
TCustomName = array[0..27]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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

You can name any memory block via MemLeaksName 375 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 375
- IsMemLeaksEnabled Function 358

10.15 TDumpCompareFunc

Comparator for memory blocks.

Unit
EMemLeaks 96

Syntax

Code (Delphi)

```
TDumpCompareFunc = function(
   const AL, AR: PPointerRecord 480
): 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 292 function.

See also
- DumpAllocationsToFile Function 292

10.16 TDumpFilterFunc

Filter for memory blocks.

Unit
EMemLeaks 96

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
TDumpFilterFunc = function(
   const ABlock: PPointerRecord 480
): 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 292 function.

See also

- DumpAllocationsToFile Function 292

10.17 TELEvAttachedFilesRequestMeth

Method event handler for OnAttachedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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 temp any 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration 392
- Procedural OnAttachedFilesRequest event handler 495
- OnZippedFilesRequest event handler 20
- EurekaLog component 29
- General events concept 75
- Other events 11

10.18 TELEvAttachedFilesRequestProc

Procedural event handler for OnAttachedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvAttachedFilesRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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 temp any 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 392
- Method OnAttachedFilesRequest event handler 19
- OnZippedFilesRequest event handler 525
- EurekaLog component 29

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- General events concept 75
- Other events 11

10.19 TELEvCustomBugIDMeth

Method event handler for OnCustomBugID event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomBugIDMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

See also

- Event handler registration 394
- Procedural OnCustomBugID event handler 496
- EurekaLog component 29
- General events concept 75
- Other events 11

10.20 TELEvCustomBugIDProc

Procedural event handler for OnCustomBugID event.

Unit

EEvents 75

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Code (Delphi)

```
TELEvCustomBugIDProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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 394
- Method OnCustomBugID event handler 22
- EurekaLog component 29
- General events concept 75
- Other events 11

10.21 TELEvCustomButtonClickMeth

Method event handler for OnCustomButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   ADialog: TObject;
   var ACloseDialog: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

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 25.
- 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 395
- Procedural OnCustomButtonClick event handler 498
- OnHelpButtonClick event handler 25
- EurekaLog component 29
- General events concept 75
- Other events 11

10.22 TELEvCustomButtonClickProc

Procedural event handler for OnCustomButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomButtonClickProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   ADialog: TObject;
   var ACloseDialog: Boolean;
   var ACallNextHandler: Boolean
);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 514.
- 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 395
- Method OnCustomButtonClick event handler 24
- OnHelpButtonClick event handler 514
- EurekaLog component 29
- General events concept 75
- Other events 11

10.23 TELEvCustomDataRequestMeth

Method event handler for OnCustomDataRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomDataRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   ALogBuilder: TObject;
   ADataFields: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Parameters

AExceptionInfo [in]
Information about exception.

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

ADataFields [out]
Custom fields to add into bug report.

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

See also

- Event handler registration 396
- Procedural OnCustomDataRequest event handler 500
- EurekaLog component 29
- General events concept 75
- Other events 11

10.24 TELEvCustomDataRequestProc

Procedural event handler for OnCustomDataRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomDataRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration 396
- Method OnCustomDataRequest event handler 23
- EurekaLog component 29
- General events concept 75
- Other events 11

10.25 TELEvCustomFileNameMeth

Method event handler for OnCustomFileName event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomFileNameMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AFileType: TEurekaLogFileType 531;
   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.

See also

- Event handler registration 397
- Procedural OnCustomFileName event handler 501
- EurekaLog component 29
- General events concept 75
- Other events 11

10.26 TELEvCustomFileNameProc

Procedural event handler for OnCustomFileName event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomFileNameProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AFileType: TEurekaLogFileType 531;
   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 397
- Method OnCustomFileName event handler 21
- EurekaLog component 29
- General events concept 75
- Other events 11

10.27 TELEvCustomWebFieldsRequestMeth

Method event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 398
- Procedural OnCustomWebFieldsRequest event handler 503
- EurekaLog component 29
- General events concept 75
- Other events 11

10.28 TELEvCustomWebFieldsRequestProc

Procedural event handler for OnCustomWebFieldsRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvCustomWebFieldsRequestProc = procedure(
   const ACustom: Pointer;
   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.

ASender [in]
Current EurekaLog options.

AWebFields [in]
Information about exception.

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

See also

- Event handler registration 398
- Method OnCustomWebFieldsRequest event handler 23

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EurekaLog component 29
- General events concept 75
- Other events 11

10.29 TELEvExceptionActionMeth

Method event handler for OnExceptionAction event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionActionMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
   const AAdditionalInfo: String;
   var AExecute: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AEurekaAction [in]
Action to perform.

AAdditionalInfo [in]
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 successfull, `False` - action was unsuccessfull;

> **Note**
>
> This argument is ignored for atSendCancelled 529.

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

> **Tip**
>
> Set to `False`, if you set 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-

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

empty value for atSending 529 and atSent 529 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 37

Note

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;
```

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'); // - OnRaise event
...
// - OnExceptionNotify will be called
// - set of OnExceptionAction events will be called
// one event for each action (saving, sending, etc).
```

See also

- Event handler registration 400
- Procedural OnExceptionAction event handler 505
- EurekaLog component 29
- General events concept 75
- Other events 11

10.30 TELEvExceptionActionProc

Procedural event handler for OnExceptionAction event.

Unit

EEvents 75

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TELEvExceptionActionProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
   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]
Information about exception.

AEurekaAction [in]
Action to perform.

AAdditionalInfo [in]
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 successfull, `False` - action was unsuccessfull;

> **Note**
>
> This argument is ignored for atSendCancelled 529.

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

> **Tip**
>
> Set to `False`, if you set 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 529 and atSent 529 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 37.

> **Note**

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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;
```

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'); // - OnRaise event
...
// - OnExceptionNotify will be called
// - set of OnExceptionAction events will be called
// one event for each action (saving, sending, etc).
```

See also

- Event handler registration 400
- Method OnExceptionAction event handler 15
- EurekaLog component 29
- General events concept 75
- Other events 11

10.31 TELEvExceptionErrorMeth

Method event handler for OnExceptionError event.

Unit

EEvents 75

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
TELEvExceptionErrorMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
   const AAdditionalInfo: String;
   var ARetry: Boolean;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

AEurekaAction [in]
Failed action.

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

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.

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.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 529 and atSent 529 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 37.

See OnExceptionAction 15 for more information.

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.

See also

- Event handler registration 401
- Procedural OnExceptionError event handler 508
- EurekaLog component 29
- General events concept 75
- Other events 11

10.32 TELEvExceptionErrorProc

Procedural event handler for OnExceptionError event.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionErrorProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   const AEurekaAction: TEurekaActionType 526;
   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]
Information about exception.

AEurekaAction [in]
Failed action.

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

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.

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.

`AAdditionalInfo` argument holds additional information for actions. Currently this argument have non-empty value for atSending 529 and atSent 529 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 37.

See OnExceptionAction 505 for more information.

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- Event handler registration 401
- Method OnExceptionError event handler 17
- EurekaLog component 29
- General events concept 75
- Other events 11

10.33 TELEvExceptionNotifyMeth

Method event handler for OnExceptionNotify event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvExceptionNotifyMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   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

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).

Tip

You can also use exception filters instead of this event.

Use OnExceptionAction 15 event to fine-control processing of exceptions.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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
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 handled b
```

See this article for example of registering event handlers.

Sample event handler may look like this:

Code (Delphi)

```
procedure MyObject.OnExceptionNotify(AExceptionInfo: TEurekaExceptionInf
begin
    // Disable exception processing by EurekaLog
    Handle := False;
    ACallNextHandler := False;
end;
```

See also

- Event handler registration 402
- Procedural OnExceptionNotify event handler 511
- EurekaLog component 29
- General events concept 75
- Other events 11

10.34 TELEvExceptionNotifyProc

Procedural event handler for OnExceptionNotify event.

Unit

EEvents 75

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Syntax

Code (Delphi)

```
TELEvExceptionNotifyProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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

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).

> **Tip**
>
> You can also use exception filters instead of this event.

Use OnExceptionAction 505 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.

© 2001-2015, 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 75 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 402
- Method OnExceptionNotify event handler 14
- EurekaLog component 29
- General events concept 75
- Other events 11

10.35 TELEvHelpButtonClickMeth

Method event handler for OnHelpButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvHelpButtonClickMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   ADialog: TObject;
   var ACallNextHandler: Boolean
```

© 2001-2015, 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 24.

See also

- Event handler registration 403
- Procedural OnHelpButtonClick event handler 514
- OnCustomButtonClick event handler 24
- EurekaLog component 29
- General events concept 75
- Other events 11
- RaiseExpected routine 389

10.36 TELEvHelpButtonClickProc

Procedural event handler for OnHelpButtonClick event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvHelpButtonClickProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   ADialog: TObject;
   var ACallNextHandler: Boolean
);
```

Parameters

ACustom [in, optional]

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 498.

See also

- Event handler registration 403
- Method OnHelpButtonClick event handler 25
- OnCustomButtonClick event handler 498
- EurekaLog component 29
- General events concept 75
- Other events 11
- RaiseExpected routine 389

10.37 TELEvPasswordRequestMeth

Method event handler for OnPasswordRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

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

Parameters

AModule [in]
Module handle which password is retrieved.

APassword [out]
Password for debug information.

© 2001-2015, 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

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 values. 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.

See also

- Event handler registration 404
- Method OnPasswordRequest event handler 516
- EurekaLog component 29
- General events concept 75
- Other events 11

10.38 TELEvPasswordRequestProc

Procedural event handler for OnPasswordRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvPasswordRequestProc = procedure(
   const ACustom: Pointer;
   const AModule: HMODULE;
   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.

APassword [out]
Password for debug information.

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 values. 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.

See also

- Event handler registration 404
- Method OnPasswordRequest event handler 18
- EurekaLog component 29
- General events concept 75
- Other events 11

10.39 TELEvRaiseMeth

Method event handler for OnRaise event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvRaiseMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 75 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 405
- Procedural OnRaise event handler 518
- EurekaLog component 29
- General events concept 75
- Other events 11

10.40 TELEvRaiseProc

Procedural event handler for OnRaise event.

Unit

EEvents 75

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
TELEvRaiseProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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 511 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 75 for example of registering event handlers.

Sample event handler may look like this:

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 405
- Method OnRaise event handler 12
- EurekaLog component 29
- General events concept 75
- Other events 11

10.41 TELEvReportGenMeth

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

EEvents 75

Syntax

Code (Delphi)

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

Parameters

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 132 instance. You need to use type cast.

Note

A more convenient case of this type 521 is declared in ELogBuilder 90 unit. Use it instead of this type whenever possible. Type from ELogBuilder doesn't require you to use typecast.

See also

- Event handler registration
- Procedural OnReportBeginGen/OnReportEndGen event handler 17

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- EurekaLog component 29
- General events concept 75
- Other events 11

10.42 TELEvReportGenMeth

Method event handler for OnReportBeginGen/OnReportEndGen events.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
TELEvReportGenMeth = procedure(
   ALogBuilder: TBaseLogBuilder 132;
   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 521
- EurekaLog component 29
- General events concept 75
- Other events 11

10.43 TELEvReportGenProc

Procedural event handler for OnReportBeginGen/OnReportEndGen event.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
TELEvReportGenProc = procedure(
   const ACustom: Pointer;
   ALogBuilder: TBaseLogBuilder 132;
   var CallNextHandler: Boolean
);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 521
- EurekaLog component 29
- General events concept 75
- Other events 11

10.44 TELEvReportGenProc

Procedural event handler for OnReportBeginGen/OnReportEndGen event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvReportGenProc = procedure(
   const ACustom: Pointer;
   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.

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 132 instance. You need to use type cast.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Note

A more convenient case of this type 521 is declared in ELogBuilder 90 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 26
- EurekaLog component 29
- General events concept 75
- Other events 11

10.45 TELEvReproduceRequestMeth

Method event handler for OnReproduceRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvReproduceRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   var AReproduceText: String;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 406
- Procedural OnReproduceRequest event handler 523
- EurekaLog component 29
- General events concept 75
- Other events 11

10.46 TELEvReproduceRequestProc

Procedural event handler for OnReproduceRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
TELEvReproduceRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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

See also

- Event handler registration 406
- Method OnReproduceRequest event handler 21
- EurekaLog component 29
- General events concept 75
- Other events 11

10.47 TELEvZippedFilesRequestMeth

Method event handler for OnZippedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestMeth = procedure(
   AExceptionInfo: TEurekaExceptionInfo 157;
   const ATempFolder: String;
   AAttachedFiles: TStrings;
   var ACallNextHandler: Boolean
) of object;
```

Parameters

AExceptionInfo [in]
Information about exception.

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

See also

- Event handler registration 407
- Procedural OnZippedFilesRequest event handler 525
- OnAttachedFilesRequest event handler 19
- EurekaLog component 29

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- General events concept 75
- Other events 11

10.48 TELEvZippedFilesRequestProc

Procedural event handler for OnZippedFilesRequest event.

Unit

EEvents 75

Syntax

Code (Delphi)

```
TELEvZippedFilesRequestProc = procedure(
   const ACustom: Pointer;
   AExceptionInfo: TEurekaExceptionInfo 157;
   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.

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

See also

- Event handler registration 407
- Method OnZippedFilesRequest event handler 20
- OnAttachedFilesRequest event handler 495
- EurekaLog component 29
- General events concept 75
- Other events 11

10.49 TEncryptBuffer

Contains an arbitrary array of bytes.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TEncryptBuffer = EWinCrypt.DATA_BLOB;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 73 unit will allocate `TEncryptBuffer` buffers via AllocMem function. Such allocations can be released via EncryptBufferFree 295 function.

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 295

10.50 TEurekaActionType

Action kind for exception processing.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit

ETypes 118

Syntax

Code (Delphi)

```
TEurekaActionType = (
    atShowingExceptionInfo 530,
    atShowedExceptionInfo 530,
    atSavingLogFile 529,
    atSavedLogFile 529,
    atSending 529,
    atSent 529,
    atSendCancelled 529,
    atTerminating 530,
    atRestarting 528,
    atCollectingExceptionInfo 528,
    atCollectedExceptionInfo 528
);
```

Members

atCollectedExceptionInfo 528	[after] Fired after collecting information about exception (building call stack)
atCollectingExceptionInfo 528	[before] Fired before collecting information about exception (building call stack)
atRestarting 528	[before] Fired before application restarting
atSavedLogFile 529	[after] Fired after saving to bug report
atSavingLogFile 529	[before] Fired before saving to bug report
atSendCancelled 529	[notify] Fired when sending was cancelled
atSending 529	[before] Fired before sending bug report
atSent 529	[after] Fired after sending bug report
atShowedExceptionInfo 530	[after] Fired after showing error dialog
atShowingExceptionInfo 530	[before] Fired before showing error dialog
atTerminating 530	[before] Fired before application termination

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This is type of action which is used in OnExceptionAction 15 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 75
- OnExceptionAction event 15

10.50.1 atCollectedExceptionInfo

[after] Fired after collecting information about exception (building call stack)

Unit

ETypes 118

See also

- TEurekaActionType Class 526

10.50.2 atCollectingExceptionInfo

[before] Fired before collecting information about exception (building call stack)

Unit

ETypes 118

See also

- TEurekaActionType Class 526

10.50.3 atRestarting

[before] Fired before application restarting

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.4 atSavedLogFile

[after] Fired after saving to bug report

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.5 atSavingLogFile

[before] Fired before saving to bug report

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.6 atSendCancelled

[notify] Fired when sending was cancelled

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.7 atSending

[before] Fired before sending bug report

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.8 atSent

[after] Fired after sending bug report

Unit
ETypes 118

See also
- TEurekaActionType Class 526

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.50.9 atShowedExceptionInfo

[after] Fired after showing error dialog

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.10 atShowingExceptionInfo

[before] Fired before showing error dialog

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.50.11 atTerminating

[before] Fired before application termination

Unit
ETypes 118

See also
- TEurekaActionType Class 526

10.51 TEurekaCallStack

Alias for TEurekaStackListV7.

Unit
ECallStack 54

Syntax

Code (Delphi)

```
TEurekaCallStack = TEurekaStackList 534;
```

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 534.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.52 TEurekaCallStackClass

Class of TEurekaBaseStackList.

Unit

ECallStack 54

Syntax

Code (Delphi)

```
TEurekaCallStackClass = class of TEurekaBaseStackList;
```

10.53 TEurekaLogFileType

Type of file name.

Unit

ETypes 118

Syntax

Code (Delphi)

```
TEurekaLogFileType = (
    ftSavedBugReport 533,
    ftBugReport 532,
    ftZIP 533,
    ftZIPCopy 533,
    ftXML 533,
    ftScreenshot 533,
    ftHTMLPage 532,
    ftMiniDump 532,
    ftLog 532
);
```

Members

ftBugReport 532	Actual working copy of bug report (located in temp folder). This bug report will be used for sending.
ftHTMLPage 532	HTML page file name.
ftLog 532	Log file name.
ftMiniDump 532	Minidump file name.
ftSavedBugReport 533	Saved bug report in its final location (as set in options). E.g. local bug report.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ftScreenshot 533	Screenshot file name.
ftXML 533	Name of XML bug report.
ftZIP 533	Packed bug report for sending (ZIP/ELP, including additional files, screenshot, etc.).
ftZIPCopy 533	File name for "Save copy in case of send failure" option.

Remarks

This type is used in OnCustomFileName event handlers to specify kind of file name to modify.

10.53.1 ftBugReport

Actual working copy of bug report (located in temp folder). This bug report will be used for sending.

Unit

ETypes 118

See also

- TEurekaLogFileType Class 531

10.53.2 ftHTMLPage

HTML page file name.

Unit

ETypes 118

See also

- TEurekaLogFileType Class 531

10.53.3 ftLog

Log file name.

Unit

ETypes 118

See also

- TEurekaLogFileType Class 531

10.53.4 ftMiniDump

Minidump file name.

Unit

ETypes 118

See also

- TEurekaLogFileType Class 531

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.53.5 ftSavedBugReport

Saved bug report in its final location (as set in options). E.g. local bug report.

Unit
ETypes 118

See also
- TEurekaLogFileType Class 531

10.53.6 ftScreenshot

Screenshot file name.

Unit
ETypes 118

See also
- TEurekaLogFileType Class 531

10.53.7 ftXML

Name of XML bug report.

Unit
ETypes 118

See also
- TEurekaLogFileType Class 531

10.53.8 ftZIP

Packed bug report for sending (ZIP/ELP, including additional files, screenshot, etc.).

Unit
ETypes 118

See also
- TEurekaLogFileType Class 531

10.53.9 ftZIPCopy

File name for "Save copy in case of send failure" option.

Unit
ETypes 118

See also
- TEurekaLogFileType Class 531

10.54 TEurekaModuleType

Alias for TModuleType 543

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
ETypes 118

Syntax

Code (Delphi)
```
TEurekaModuleType = TModuleType 543;
```

Remarks
See description of TModuleType 543.

See also
- TModuleType

10.55 TEurekaStackList

Alias for TEurekaStackListV7.

Unit
ECallStack 54

Syntax

Code (Delphi)
```
TEurekaStackList = TEurekaStackListV7 189;
```

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 530.

10.56 TExceptionNotifyMode

Specifies work mode of exception manager.

Unit
EExceptionManager 84

Syntax

Code (Delphi)
```
TExceptionNotifyMode = (
    emUnknown 536,
    emFull 536,
    emD2009 535,
    emHook 536,
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
    emNull 536
);
```

Members

emD2009 535	Extended memory manager is not set. Hi-level hook was set for Exception class. (Available only for Delphi/C++ Builder 2009 and later.)
emFull 536	Extended memory manager was set. Full features.
emHook 536	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 536	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.
emUnknown 536	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 203. 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 216
- TExceptionManager class 203

10.56.1 emD2009

Extended memory manager is not set. Hi-level hook was set for Exception class. (Available only for Delphi/C++ Builder 2009 and later.)

Unit

EExceptionManager 84

See also

- TExceptionNotifyMode Class 534

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.56.2 emFull

Extended memory manager was set. Full features.

Unit

EExceptionManager 84

See also

- TExceptionNotifyMode Class 534

10.56.3 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 84

See also

- TExceptionNotifyMode Class 534

10.56.4 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.

Unit

EExceptionManager 84

See also

- TExceptionNotifyMode Class 534

10.56.5 emUnknown

Work mode is not set. Exception manager was not initialized yet.

Unit

EExceptionManager 84

See also

- TExceptionNotifyMode Class 534

10.57 TFreezeThreadClass

Class of freeze detection thread.

Unit

EFreeze 86

Syntax

Code (Delphi)

```
TFreezeThreadClass = class of TFreezeThread;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Remarks

This type accepts TFreezeThread 217 class or any of its descendants.

This type is used in FreezeThreadClass 585 global variable.

Caution!

`TFreezeThread` is abstract class, you should not use it. Use TMainThreadFreezeDetectionThread 224 or TWCTFreezeDetectionThread 234 or implement your own freeze detection class.

See also

- TFreezeThread Class 217
- FreezeThreadClass Variable 585
- TMainThreadFreezeDetectionThread Class 224
- TWCTFreezeDetectionThread Class 234
- InitCheckFreeze Function 337

10.58 TInstruction

A CPU instruction.

Unit

EDisAsm 70

Syntax

Code (Delphi)

```
TInstruction = record
    Start 539: PAddress;
    Next 538: PAddress;
    Accessible 537: Boolean;
    Valid 539: Boolean;
    PlatformSpecific 538: 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 287
- InstructionToString Function 341
- PInstruction Type 480

10.58.1 Accessible

Is instruction readable?

Unit

EDisAsm 70

© 2001-2015, 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 539 address is readable. You will get Access Violation exception if you try to read from `Start` when Accessible if `False`.

See also

- Start Field 539
- Next Field 538
- Valid Field 539
- TInstruction Class 537

10.58.2 Next

Next CPU instruction.

Unit

EDisAsm 70

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 539 from `Next`.

See also

- Start Field 539
- Accessible Field 537
- Valid Field 539
- TInstruction Class 537

10.58.3 PlatformSpecific

Other platform-specific fields.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EDisAsm 70

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 539
- Next Field 538
- Accessible Field 537
- Valid Field 539
- TInstruction Class 537

10.58.4 Start

Start of the CPU instruction.

Unit
EDisAsm 70

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 538
- Accessible Field 537
- Valid Field 539
- TInstruction Class 537

10.58.5 Valid

Is instruction valid?

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EDisAsm 70

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 539
- Next Field 538
- Accessible Field 537
- TInstruction Class 537

10.59 TMD5Hash

MD5 hash.

Unit
EHash 87

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 374

10.60 TMemoryError

Type of detected memory error.

Unit
EMemLeaks 96

Syntax

```
Code (Delphi)
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
TMemoryError = (
    leNone 542,
    leOverflow 543,
    leMultiFree 542,
    leInvalidBlock 542,
    leHang 542,
    leUseAfterFree 543,
    leOutOfMemory 542
);
```

Members

leHang 542	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 542	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 542	Double-free of memory block or object.
leNone 542	No memory error is detected.
leOutOfMemory 542	There is not enough memory to allocate requested amount of memory in GetMem or AllocMem.
leOverflow 543	Buffer overflow is detected. Some code damaged/overwrote internal EurekaLog's headers.
leUseAfterFree 543	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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.60.1 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.

Unit

EMemLeaks 96

See also

- TMemoryError Class 540

10.60.2 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

EMemLeaks 96

See also

- TMemoryError Class 540

10.60.3 leMultiFree

Double-free of memory block or object.

Unit

EMemLeaks 96

See also

- TMemoryError Class 540

10.60.4 leNone

No memory error is detected.

Unit

EMemLeaks 96

See also

- TMemoryError Class 540

10.60.5 leOutOfMemory

There is not enough memory to allocate requested amount of memory in GetMem or AllocMem.

Unit

EMemLeaks 96

See also

- TMemoryError Class 540

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.60.6 leOverflow

Buffer overflow is detected. Some code damaged/overwrote internal EurekaLog's headers.

Unit
EMemLeaks 96

See also
- TMemoryError Class 540

10.60.7 leUseAfterFree

Detected attempt to use already released object or interface. A virtual method was called on already deleted object.

Unit
EMemLeaks 96

See also
- TMemoryError Class 540

10.61 TMemoryErrors

Defines set of possible memory errors.

Unit
EMemLeaks 96

Syntax

Code (Delphi)
```
TMemoryErrors = set of TMemoryError;
```

Remarks
This type is used in MemLeaksErrorsToIgnore global variable.

See also
- TMemoryError Type 540
- MemLeaksErrorsToIgnore Global Variable

10.62 TModuleType

Indicates kind of executable module.

Unit
ETypes 118

Syntax

Code (Delphi)
```
TModuleType = (
    mtUnknown 545,
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
    mtProgram 544,
    mtLibrary 544,
    mtPackage 544
);
```

Members

mtLibrary 544	.dll file
mtPackage 544	.bpl file
mtProgram 544	.exe file
mtUnknown 545	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 327
- TEurekaModulesList.FindModuleType
- TDesignModule.GetModuleType

10.62.1 mtLibrary

.dll file

Unit

ETypes 118

See also

- TModuleType Class 543

10.62.2 mtPackage

.bpl file

Unit

ETypes 118

See also

- TModuleType Class 543

10.62.3 mtProgram

.exe file

Unit

ETypes 118

See also

- TModuleType Class 543

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.62.4 mtUnknown

Module type wasn't recognized

Unit
ETypes 118

See also
- TModuleType Class 543

10.63 TPointerRecord

Header for memory blocks allocated by EurekaLog.

Unit
EMemLeaks 96

Syntax

Code (Delphi)

```
TPointerRecord = packed record
   Prev, Next: PPointerRecord 480;
   Module: HMODULE;
   Tag: Pointer;
   OrigSize: PtrUInt;
   SingleSize 547: PtrUInt;
   TotalSize: PtrUInt;
   Name: TCustomName 492;
   Parent: Pointer;
   BlockState 546: TBlockState;
   BlockType 546: TBlockType 483;
   Count: Word;
   Flags: DWORD;
   Reserved1: DWORD;
   Reserved2: DWORD;
   CallStack 546: 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 292
- TDumpCompareFunc Type 493
- TDumpFilterFunc Type 493
- PPointerRecord Type 480

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.63.1 BlockState

State of memory block.

Unit
EMemLeaks 96

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 Class 545

10.63.2 BlockType

Type of memory block.

Unit
EMemLeaks 96

Syntax

Code (Delphi)
```
BlockType: TBlockType 483;
```

Value
Indicate type of the block.

Remarks
This value represents type of memory block (object, array, string or RAW).

See also
- TPointerRecord Class 545

10.63.3 CallStack

Call stack of allocation.

Unit
EMemLeaks 96

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 Class 545

10.63.4 SingleSize

Memory block size.

Unit
EMemLeaks 96

Syntax

```
Code (Delphi)
SingleSize: PtrUInt;
```

Value
Actual size of the block, in bytes.

Remarks
This value represents size of the memory block in bytes.

See also
- TPointerRecord Class 545

10.64 TResponse

Result of exception's processing or sending bug report.

Unit
ETypes 118

Syntax

```
Code (Delphi)
TResponse = packed record
    SendResult 549: TSendResult 555;
    IssueID 549: Int64;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
    ErrorCode 548: Integer;
    ErrorMessage 548: String;
    SuccessMessage 550: 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 205

10.64.1 ErrorCode

Error code for failed operations.

Unit

ETypes 118

Syntax

Code (Delphi)

```
ErrorCode: Integer;
```

Value

System or any other error code. Numeric value. May be 0, if no error code available.

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 Class 547

10.64.2 ErrorMessage

Error message (failed operations only).

Unit

ETypes 118

Syntax

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
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 Class 547

10.64.3 IssueID

Ticket/issue ID in bug tracker.

Unit
ETypes 118

Syntax

```
Code (Delphi)
IssueID: Int64;
```

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. It can be `Integer` or `Int64`.

This value can be set only for succeeded operations.

See also
- TResponse Class 547

10.64.4 SendResult

Status of operation.

Unit
ETypes 118

Syntax

```
Code (Delphi)
SendResult: TSendResult 555;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Value

HRESULT value indicating success/failure status of operation. See TSendResult 555 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 555
- TResponse Class 547

10.64.5 SuccessMessage

Success message (succeeded operations only).

Unit

ETypes 118

Syntax

Code (Delphi)

```
SuccessMessage: String;
```

Value

Success message in arbitrary format. This message may also be a URL.

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 Class 547

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.65 TRSAExport

Defines how to export/import RSA key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TRSAExport = (
    rsBLOB 551,
    rsDER 552,
    rsPEM 552
);
```

Members

rsBLOB 551	Export/import in RAW BLOB with header (Microsoft PUBLICKEYBLOB/ PRIVATEKEYBLOB, little-endian byte order).
rsDER 552	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 552	Export/import in text form (PEM=base64-encoded DER).

See also

- RSALoadPublicKey 419
- RSALoadPrivateKey 418
- RSASavePublicKey 421
- RSASavePrivateKey 420

10.65.1 rsBLOB

Export/import in RAW BLOB with header (Microsoft PUBLICKEYBLOB/PRIVATEKEYBLOB, little-endian byte order).

Unit

EEncrypt 73

See also

- TRSAExport Class 551

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.65.2 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 73

See also
- TRSAExport Class 551

10.65.3 rsPEM

Export/import in text form (PEM = base64-encoded DER).

Unit
EEncrypt 73

See also
- TRSAExport Class 551

10.66 TRSAKey

Key for RSA encryption.

Unit
EEncrypt 73

Syntax

Code (Delphi)

```
TRSAKey = record
    PublicKey 553: TEncryptBuffer 525;
    PrivateKey 553: TEncryptBuffer 525;
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 414 function.

This key can be created by RSAGenKey 417 function or be imported by RSALoadPublicKey 419 and RSALoadPrivateKey 418 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 414 function to encrypt (by using your public key) "clear text" data, use RSADecrypt 411 function to decrypt (by using your private key) encrypted data and get "clear text" back.

Use RSASign 422 function to sign (by using your private key) your data, use RSAVerify 424 to verify (by using your public key) a signature.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- RSAEncrypt Function 414
- RSADecrypt Function 411
- RSASign Function 422
- RSAVerify Function 424
- RSALoadPublicKey Function 419
- RSALoadPrivateKey Function 418
- RSADestroyKey Function 414

10.66.1 PrivateKey

Contains private key part of RSA key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
PrivateKey: TEncryptBuffer 525;
```

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 420 function to export this value. Do not copy this value directly, as it will be unusable.

See also

- RSASavePrivateKey Function 420
- TRSAKey Class 552

10.66.2 PublicKey

Contains public key part of RSA key.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
PublicKey: TEncryptBuffer 525;
```

Value

RAW binary data of public key.

Remarks

This field holds unprotected public key in clear form.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Use RSASavePublicKey 421 function to export this value or simply copy this value directly.

See also

- RSASavePublicKey Function 421
- TRSAKey Class 552

10.67 TRSASignature

Digital signature.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TRSASignature = TEncryptBuffer 525;
```

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 428 function to release this type.

See also

- RSASign Function 422
- RSAVerify Function 424
- TEncryptBuffer Type 525

10.68 TSalt

Salt value for deriving key from password.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TSalt = TTEAKey 567;
```

Remarks

Use InitSalt 338 to obtain new random salt. Pass the obtained value to TEADeriveKey 443 function or TwofishDeriveKey 455 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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- PSalt Type 481
- TEADeriveKey Function 443
- TwofishDeriveKey Function 455
- InitSalt Function 338

10.69 TSDBMHash

SDBM hash.

Unit
EHash 87

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 427

10.70 TSendResult

Result kind for exception processing.

Unit
ETypes 118

Syntax

Code (Delphi)
```
TSendResult = (
    srSent 559 = S_OK,
    srBugClosed 557 = S_FALSE,
    srNoExceptionInfo 559 = S_NONE,
    srRestart 559 = E_RESTART,
    srCancelled 557 = E_CANCELLED,
    srUnknownError 560 = E_UNEXPECTED,
    srUnsupportedSender 560 = E_NOINTERFACE,
    srConnectionError 557 = E_CONNECTION_REFUSED,
    srObsoleteVersion 559 = E_OBSOLETE,
    srInvalidLogin 558 = E_LOGON_FAILED,
    srInvalidSearch 558 = E_INVALID_SEARCH,
    srInvalidSelection 559 = E_INVALID_SELECTION,
    srInvalidRead 558 = E_INVALID_READ,
    srInvalidInsert 558 = E_INVALID_INSERT,
    srInvalidModify 558 = E_INVALID_MODIFY
);
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Members

srBugClosed 557	[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.
srCancelled 557	[FAILURE] Exception processing was cancelled by user via dialog.
srConnectionError 557	[FAILURE] Network connection error. This includes DNS errors.
srInvalidInsert 558	[FAILURE] There was error when creating (opening) new case (ticket). For non-bug trackers methods this means error during content sending.
srInvalidLogin 558	[FAILURE] There was error during log in stage. Most common reason is wrong login/password pair.
srInvalidModify 558	[FAILURE] There was error when modifying existing case (ticket). Applicable only for bug trackers.
srInvalidRead 558	[FAILURE] There was error during reading project or case options. Possible access denied error?
srInvalidSearch 558	[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 559	[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.
srNoExceptionInfo 559	[SUCCESS] There is no active exception. Nothing to do.
srObsoleteVersion 559	[FAILURE] Project version is obsolete. Bug reports are no longer required, submission is closed.
srRestart 559	[FAILURE] Exception processing must be restarted. This value is typically used to switch between dialogs and handlers.
srSent 559	[SUCCESS] Exception was processed

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

	without errors. If sending is set up - then bug report was sent succcessfully.
srUnknownError 560	[FAILURE] Common error code. It's used when no other code can be used.
srUnsupportedSender 560	[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 205

10.70.1 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 118

See also

- TSendResult Class 555

10.70.2 srCancelled

[FAILURE] Exception processing was cancelled by user via dialog.

Unit

ETypes 118

See also

- TSendResult Class 555

10.70.3 srConnectionError

[FAILURE] Network connection error. This includes DNS errors.

Unit

ETypes 118

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- TSendResult Class 555

10.70.4 srInvalidInsert

[FAILURE] There was error when creating (opening) new case (ticket). For non-bug trackers methods this means error during content sending.

Unit

ETypes 118

See also

- TSendResult Class 555

10.70.5 srInvalidLogin

[FAILURE] There was error during log in stage. Most common reason is wrong login/password pair.

Unit

ETypes 118

See also

- TSendResult Class 555

10.70.6 srInvalidModify

[FAILURE] There was error when modifying existing case (ticket). Applicable only for bug trackers.

Unit

ETypes 118

See also

- TSendResult Class 555

10.70.7 srInvalidRead

[FAILURE] There was error during reading project or case options. Possible access denied error?

Unit

ETypes 118

See also

- TSendResult Class 555

10.70.8 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 118

See also

- TSendResult Class 555

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.70.9 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 118

See also
- TSendResult Class 555

10.70.10 srNoExceptionInfo

[SUCCESS] There is no active exception. Nothing to do.

Unit
ETypes 118

See also
- TSendResult Class 555

10.70.11 srObsoleteVersion

[FAILURE] Project version is obsolete. Bug reports are no longer required, submission is closed.

Unit
ETypes 118

See also
- TSendResult Class 555

10.70.12 srRestart

[FAILURE] Exception processing must be restarted. This value is typically used to switch between dialogs and handlers.

Unit
ETypes 118

See also
- TSendResult Class 555

10.70.13 srSent

[SUCCESS] Exception was processed without errors. If sending is set up - then bug report was sent succcessfully.

Unit
ETypes 118

See also
- TSendResult Class 555

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.70.14 srUnknownError

[FAILURE] Common error code. It's used when no other code can be used.

Unit
ETypes 118

See also
- TSendResult Class 555

10.70.15 srUnsupportedSender

[FAILURE] Unknown sender class. It's either wrong class name specified or sender class wasn't registered.

Unit
ETypes 118

See also
- TSendResult Class 555

10.71 TSHA1Hash

SHA-1 hash.

Unit
EHash 87

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 434

10.72 TSHA256Hash

SHA-2/SHA-256 hash.

Unit
EHash 87

Syntax

Code (Delphi)
```
TSHA256Hash = packed array[0..31]of Byte;
```

© 2001-2015, 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 434

10.73 TSignatureInfo

Information about digital signature of executable file.

Unit

ESignature 112

Syntax

Code (Delphi)

```
TSignatureInfo = record
    FileName 562: String;
    Signed 564: Boolean;
    ProgramName 563: String;
    PublisherLink 563: String;
    MoreInfoLink 562: String;
    Certificate 561: TCertificateInfo 485;
    Timestamped 566: Boolean;
    Timestamp 565: TDateTime;
    TimestampCertificate 565: TCertificateInfo 485;
end;
```

Remarks

This record contains information about digital signature (and its certificates) of executable file passed to SignatureGetInfo 436 function.

FileName 562 and Signed 564 fields are always valid. All other fields are valid only if Signed 564 field is `True`. Additionally, Timestamp 565 and TimestampCertificate 565 fields are valid only if Timestamped 566 field is `True`.

See also

- SignatureGetInfo Function 436

10.73.1 Certificate

Signature's digital certificate.

Unit

ESignature 112

Syntax

Code (Delphi)

```
Certificate: TCertificateInfo 485;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 564 field is `True`.

See also
- TimestampCertificate Field 565
- Signed Field 564
- TSignatureInfo Class 561

10.73.2 FileName

Original file name of executable.

Unit
ESignature 112

Syntax

Code (Delphi)

```
FileName: String;
```

Value
Executable file name.

Remarks
This is the same value as `AFileName` argument passed to SignatureGetInfo 436. This field is always valid.

Examples
```
'C:\Projects\MyApp.exe'
```

See also
- SignatureGetInfo Function 436
- TSignatureInfo Class 561

10.73.3 MoreInfoLink

Additional link.

Unit
ESignature 112

Syntax

Code (Delphi)

```
MoreInfoLink: String;
```

© 2001-2015, 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 564 field is `True`.

Examples
```
'http://www.eurekalog.com/'
```

See also
- PublisherLink Field 563
- Signed Field 564
- TSignatureInfo Class 561

10.73.4 ProgramName

Program name.

Unit
ESignature 112

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 564 field is `True`.

Examples
```
'EurekaLog Tools Pack Installer'
```

See also
- Signed Field 564
- TSignatureInfo Class 561

10.73.5 PublisherLink

Publisher link.

Unit
ESignature 112

© 2001-2015, 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 564 field is `True`.

Examples

```
'http://www.eurekalog.com/'
```

See also

- MoreInfoLink Field 562
- Signed Field 564
- TSignatureInfo Class 561

10.73.6 Signed

Indicate if file has digital signature.

Unit

ESignature 112

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 562) are valid only if this field is `True`. If Signed is `False` then other fields (except FileName 562) are empty.

See also

- FileName Field 562
- Timestamped Field 566
- TSignatureInfo Class 561

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.73.7 Timestamp

Timestamp.

Unit
ESignature 112

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 561 field.

This field is valid only if Timestamped 566 field is `True`.

Important!

Currently, SignatureGetInfo 436 is unable to retrieve info about `szOID_TIMESTAMP_TOKEN`/ `szOID_RFC3161_counterSign` (which is `'1.2.840.113549.1.9.16.1.4'`). This type of timestamp is used by Microsoft to sign its own executables.

SignatureGetInfo 436 supports only `szOID_RSA_signingTime` (`'1.2.840.113549.1.9.5'`). This type of signature is commonly used.

Examples
```
2015.10.23 14:48:02
```

See also
- TimestampCertificate Field 565
- TSignatureInfo Class 561

10.73.8 TimestampCertificate

Certificate used to sign timestamp.

Unit
ESignature 112

Syntax

Code (Delphi)

```
TimestampCertificate: TCertificateInfo 485;
```

Value
Information about certificate which was used to counter-sign the timestamp value.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 561 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 565 field.

See also

- Timestamped Field 566
- Timestamp Field 565
- Certificate Field 561
- TSignatureInfo Class 561

10.73.9 Timestamped

Indicate if signature is timestamped.

Unit

ESignature 112

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 564 field is `True`.

If this field is `True`, then Timestamp 565 field and TimestampCertificate 565 are also valid. Otherwise (`Timestamped` is `False`) Timestamp 565 and TimestampCertificate 565 fields are zero.

Important!

Currently, SignatureGetInfo 436 is unable to retrieve info about `szOID_TIMESTAMP_TOKEN`/ `szOID_RFC3161_counterSign` (which is `'1.2.840.113549.1.9.16.1.4'`). This type of timestamp is used by Microsoft to sign its own executables.

SignatureGetInfo 436 supports only `szOID_RSA_signingTime` (`'1.2.840.113549.1.9.5'`). This type of signature is commonly used.

See also

- Signed Field 564
- Timestamp Field 565
- TimestampCertificate Field 565
- SignatureGetInfo Function 436
- TSignatureInfo Class 561

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.74 TStreamConstructor

Function to Create a Compression/Decompression stream

Unit
EZip 124

Syntax

Code (Delphi)

```
TStreamConstructor = function(
   InStream: TStream;
   const ZipFile: TZipFile 235;
   const Item: TZipHeader 577
): TStream;
```

10.75 TTEAKey

Key for TEA encryption.

Unit
EEncrypt 73

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 481/TSalt 554 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 447 function once you have finished working with key - to erase it with zeros.

This key is derived from string (or binary) password via TEADeriveKey 443 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 447 function to encrypt "clear text" data, use TEADecrypt 437 function to decrypt encrypted data and get "clear text" data back.

See also
- TEADeriveKey Function 443
- TEAEncrypt Function 447
- TEADecrypt Function 437
- PSalt Type 481
- TSalt Type 554

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.76 TThreadKind

Indicates thread kind.

Unit

EThreadsManager 117

Syntax

Code (Delphi)

```
TThreadKind = (
    tkUnknown,
    tkGeneric 568,
    tkBeginThread 568,
    tkTThread 569
);
```

Members

tkBeginThread 568	RTL thread (created with BeginThread)
tkGeneric 568	Windows thread (created with CreateThread)
tkTThread 569	RTL thread (created with TThread's descendant)

Remarks

This type is used to specify thread kind in TEurekaThreadData class (see Kind property).

See also

- TEurekaThreadData Class 190
- TEurekaThreadData.Kind Property 195

10.76.1 tkBeginThread

RTL thread (created with BeginThread)

Unit

EThreadsManager 117

See also

- TThreadKind Class 568

10.76.2 tkGeneric

Windows thread (created with CreateThread)

Unit

EThreadsManager 117

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also

- TThreadKind Class 568

10.76.3 tkTThread

RTL thread (created with TThread's descendant)

Unit

EThreadsManager 117

See also

- TThreadKind Class 568

10.77 TTwofishInitVector

Initialization Vector (IV) for Twofish.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TTwofishInitVector = packed array[0..15]of Byte;
```

Remarks

Use TwofishInitIV 459 to obtain new random initialization vector. Pass the obtained value to TwofishEncrypt 457 and store it with encrypted data - to be passed into TwofishDecrypt 454 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 457
- TwofishDecrypt Function 454
- TwofishInitIV Function 459

10.78 TTwofishKey

(Precalculated) Key for Twofish encryption.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TTwofishKey = TEncryptBuffer 525;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 457 function.

This key can be created by TwofishDeriveKey 455 function or be imported by TwofishInitKey 459 function. You need to be careful when handling this key. I.e. leaking this key means leaking your key/password.

Use TwofishEncrypt 457 function to encrypt "clear text" data, use TwofishDecrypt 454 function to decrypt encrypted data and get "clear text" back.

See also

- TTwofishRAWKey Type 570
- TwofishDeriveKey Function 455
- TwofishInitKey Function 459
- TwofishEncrypt Function 457
- TwofishDecrypt Function 454
- TwofishDestroyKey Function 457

10.79 TTwofishRAWKey

(RAW) Key for Twofish encryption.

Unit

EEncrypt 73

Syntax

Code (Delphi)

```
TTwofishRAWKey = packed array[0..31]of Byte;
```

Remarks

This is RAW binary key for Twofish.

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 459 function.

See also

- TTwofishKey Type 569
- TwofishInitKey Function 459

10.80 TZipCompression

Zip compression method.

Unit

EZip 124

Syntax

Code (Delphi)

```
TZipCompression = (
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

```
    zcStored 576,
    zcShrunk 575,
    zcReduce1 574,
    zcReduce2 574,
    zcReduce3 574,
    zcReduce4 574,
    zcImplode 573,
    zcTokenize 576,
    zcDeflate 572,
    zcDeflate64 572,
    zcPKImplode 573,
    zcReserved11 574,
    zcBZIP2 572,
    zcReserved13 575,
    zcLZMA 573,
    zcReserved15 575,
    zcReserved16 575,
    zcReserved17 575,
    zcTERSE 576,
    zcLZ77 573,
    zcWavePack 576 = 97,
    zcPPMdI1 573
);
```

Members

zcBZIP2 572	File is compressed using BZIP2 algorithm
zcDeflate 572	The file is Deflated
zcDeflate64 572	Enhanced Deflating using Deflate64
zcImplode 573	The file is Imploded
zcLZ77 573	IBM LZ77 z Architecture (PFS)
zcLZMA 573	LZMA (EFS)
zcPKImplode 573	PKWARE Data Compression Library Imploding (old IBM TERSE)
zcPPMdI1 573	PPMd version I, Rev 1
zcReduce1 574	The file is Reduced with compression factor 1
zcReduce2 574	The file is Reduced with compression factor 2
zcReduce3 574	The file is Reduced with compression factor 3

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

zcReduce4 574	The file is Reduced with compression factor 4
zcReserved11 574	Reserved by PKWARE
zcReserved13 575	Reserved by PKWARE
zcReserved15 575	Reserved by PKWARE
zcReserved16 575	Reserved by PKWARE
zcReserved17 575	Reserved by PKWARE
zcShrunk 575	The file is Shrunk
zcStored 576	The file is stored (no compression)
zcTERSE 576	File is compressed using IBM TERSE (new)
zcTokenize 576	Reserved for Tokenizing compression algorithm
zcWavePack 576	WavPack compressed data

See also

- TZipCompressionToString function 460
- TZipFile class 235

10.80.1 zcBZIP2

File is compressed using BZIP2 algorithm

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.2 zcDeflate

The file is Deflated

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.3 zcDeflate64

Enhanced Deflating using Deflate64

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.4 zcImplode

The file is Imploded

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.5 zcLZ77

IBM LZ77 z Architecture (PFS)

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.6 zcLZMA

LZMA (EFS)

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.7 zcPKImplode

PKWARE Data Compression Library Imploding (old IBM TERSE)

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.8 zcPPMdI1

PPMd version I, Rev 1

Unit
EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Requirements

IDE	Delphi 6 or above; C++ Builder 6 or above

See also

- TZipCompression Class 570

10.80.9 zcReduce1

The file is Reduced with compression factor 1

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.10 zcReduce2

The file is Reduced with compression factor 2

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.11 zcReduce3

The file is Reduced with compression factor 3

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.12 zcReduce4

The file is Reduced with compression factor 4

Unit

EZip 124

See also

- TZipCompression Class 570

10.80.13 zcReserved11

Reserved by PKWARE

Unit

EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

See also
- TZipCompression Class 570

10.80.1.4 zcReserved13

Reserved by PKWARE

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.1.5 zcReserved15

Reserved by PKWARE

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.1.6 zcReserved16

Reserved by PKWARE

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.1.7 zcReserved17

Reserved by PKWARE

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.1.8 zcShrunk

The file is Shrunk

Unit
EZip 124

See also
- TZipCompression Class 570

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

10.80.19 zcStored

The file is stored (no compression)

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.20 zcTERSE

File is compressed using IBM TERSE (new)

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.21 zcTokenize

Reserved for Tokenizing compression algorithm

Unit
EZip 124

See also
- TZipCompression Class 570

10.80.22 zcWavePack

WavPack compressed data

Unit
EZip 124

Requirements

IDE	Delphi 6 or above; C++ Builder 6 or above

See also
- TZipCompression Class 570

10.81 TZipEndOfCentralHeader

Final block written to zip file

Unit
EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

10.82 TZipHeader

TZipHeader contains information about a file in a zip archive.

Unit

EZip 124

Syntax

```
Code (Delphi)
TZipHeader = packed record
   MadeByVersion 578: UInt8;
   HostOS 578: UInt8;
   RequiredVersion 579: 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;
```

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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.

10.82.1 HostOS

Version made by (HostOS byte; upper)

Unit

EZip 124

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
- 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 Class 577

10.82.2 MadeByVersion

Version made by (software byte; lower)

Unit

EZip 124

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

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 Class 577

10.82.3 RequiredVersion

Version needed to extract (2 bytes)

Unit

EZip 124

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

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- 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 should also be set. This field should be set appropriately to indicate whether Version 1 or Version 2 format is in use.

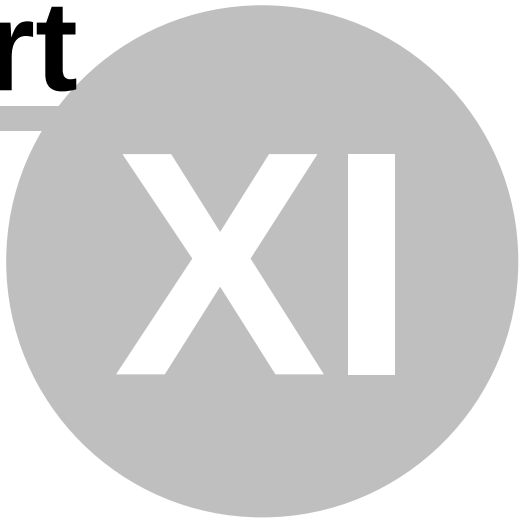
Tip
Start of Local Header

See also

- TZipHeader Class 577

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Part



11 All Globals

DefaultTracerMask 582	Mask for default call stack class.
EEurekaLogCurrentVersion 583	Current EurekaLog version (numeric).
EEurekaLogLicense 583	No description available
EEurekaLogVersion 584	Current EurekaLog version (textual).
EurekaCallStackClass 584	Default call stack class.
FreezeThreadClass 585	Determinate class to be used as background freeze detection thread.
LogBuilderClass 586	Specifies global log builder class to use.
CENTRALHEADER 586	Signature of .zip file.
DATADESCRIPTOR 587	Data descriptor signature.
LOCALHEADER 587	Local file header signature.
ZIPENDOFHEADER 588	End of central dir signature.

11.1 DefaultTracerMask

Mask for default call stack class.

Unit

ECallStack 54

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 584 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 584 as well.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Note
Default is mask for TEurekaStackListV7 189 class.

See also

- EurekaCallStackClass 584

11.2 EEurekaLogCurrentVersion

Current EurekaLog version (numeric).

Unit

EConsts 59

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 584 constant for human-readable version information.

Note
This value may be less than actual EurekaLog version.

See also

- EEurekaLogVersion 584

11.3 EEurekaLogLicense

EurekaLog's license type.

Unit

EConsts 59

Syntax

```
Code (Delphi)
EEurekaLogLicense = 'Enterprise';
```

© 2001-2015, 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 584

11.4 EEurekaLogVersion

Current EurekaLog version (textual).

Unit
EConsts 59

Syntax

```
Code (Delphi)
EEurekaLogVersion = '7.5.1.151 alpha 4';
```

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 322
- EEurekaLogCurrentVersion 583
- EEurekaLogLicense 583

11.5 EurekaCallStackClass

Default call stack class.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Unit
ECallStack 54

Syntax

Code (Delphi)
```
EurekaCallStackClass: TEurekaCallStackClass 531 = 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 331 / GetTracerClass 332 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 582 as well.

Note

Default is TEurekaStackListV7 189 class.

See also
- DefaultTracerMask 582
- GetTracer 331
- GetTracerClass 332

11.6 FreezeThreadClass

Determinate class to be used as background freeze detection thread.

Unit
EFreeze 86

Syntax

Code (Delphi)
```
FreezeThreadClass: TFreezeThreadClass 536;
```

Value
Class to create by InitCheckFreeze 337 function.

Remarks
This variable is used by InitCheckFreeze 337 function to create a new background freeze detection thread. You can use build-in classes (TMainThreadFreezeDetectionThread 224 or TWCTFreezeDetectionThread 234) or

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

implement your own freeze detection thread class. Class has to be descendant of TFreezeThread 217 and it must override Execute 218 method.

Note
You need to re-initialize freeze detection after changing this variable if freeze checks are running. Use DoneCheckFreeze 291 and InitCheckFreeze 337 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 217
- InitCheckFreeze Function 337
- DoneCheckFreeze Function 291
- TFreezeThreadClass Type 536
- TMainThreadFreezeDetectionThread Class 224
- TWCTFreezeDetectionThread Class 234

11.7 LogBuilderClass

Specifies global log builder class to use.

Unit

ELogBuilder 90

Syntax

Code (Delphi)

```
LogBuilderClass: TBaseLogBuilderClass = TLogBuilder;
```

Value

A specific log builder class. TLogBuilder 223 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.

11.8 CENTRALHEADER

Signature of .zip file.

Unit

EZip 124

Syntax

Code (Delphi)

```
SIGNATURE_CENTRALHEADER: UInt32 = $02014B50;
```

© 2001-2015, 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 577

11.9 DATADESCRIPTOR

Data descriptor signature.

Unit
EZip 124

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.

11.10 LOCALHEADER

Local file header signature.

Unit
EZip 124

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.

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

11.11 ZIPENDOFHEADER

End of central dir signature.

Unit

EZip 124

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 576.

See also

- TZipEndOfCentralHeader 576

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

Index

- A -

Accessible 537
ActiveControl 160
ActiveControlClass 160
ActiveControlText 161
ActiveForm 161
ActiveFormClass 162
ActiveFormText 162
Add 238
AdditionalInfo 162
Address 163
AllocatePageAlign 260
AllocMemCountEx 261
AllocMemCountMax 262
AllocMemSizeEx 262
AllocMemSizeMax 263
AquireThreadData 264
atCollectedExceptionInfo 528
atCollectingExceptionInfo 528
atRestarting 528
atSavedLogFile 529
atSavingLogFile 529
atSendCancelled 529
atSending 529
atSent 529
atShowedExceptionInfo 530
atShowingExceptionInfo 530
atTerminating 530
AutoHandleException 232

- B -

BaseException 163
BeginThreadEx 264
BlockState 546
BlockType 546
btData 484
btDynArray 484
btObject 484
btString 484
BugID 164
BugIDSource 164
BugIDStr 165
Build 155
BuildBugReport 267
BuildReport 133

- C -

CallerAddress 192
CallerFunction 193
CallStack 136, 165, 546
Certificate 561
CheckELMagicCode 268
CheckHeap 268
ChecksPaused 218
ClearThreadExceptions 205
Close 240
Comment 244
ConfigDelete 269
ConfigFileName 270
ConfigIsRegistry 271
ConfigKeyPath 272
ConfigReadBool 273
ConfigReadFloat 274
ConfigReadInteger 275
ConfigReadString 276
ConfigWriteBool 278
ConfigWriteFloat 279
ConfigWriteInteger 280
ConfigWriteString 281
Context 165
ConvertAddress 282
Count 201
CRC16Hash 283
CRC32Hash 284
Create 133, 153, 221, 228, 238
ctEqual 490
ctGetLastError 491
ctHResult 491
ctNone 491
ctNotEqual 491
ctNTStatus 491
CurrentEurekaLogOptions 8, 284
CurrentInstance 286
Cursor 166
CursorPos 166
CustomFields 137
CustomID 167

- D -

DecodeConfigValue 286
DecodeIdent 140
DecodeInstruction 287
DefaultTracerMask 582
Destroy 238

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

DisableMemLeaksRegistering 288
DisasmFunction 289
DisasmInstruction 290
DoneCheckFreeze 291
DumpAllocationsToFile 292

- E -

EAnsiFileNamesFix 48
EAppCLX 49
EAppCrossModule 49
EAppFMX 49
EAppHTTPApp 49
EAppType 50
EAppVCL 52
EAppWebBroker 52
EBase 52
EBaseClasses 53
ECallStack 54
EClasses 55
EClassUtils 56
ECompatibility 56
EComponent 56
EConfig 57
EConsts 59
ECore 59
EDbgHelp 60
EDebugDWARF 60
EDebugEL 60
EDebugEL7 61
EDebugExports 61
EDebugInfo 62
EDebugJCL 62
EDebugMap 62
EDebugME 63
EDebugMS 63
EDebugStabs 63
EDebugTD32 64
EDialog 64
EDialogCGI 66
EDialogConsole 66
EDialogHTMLBased 66
EDialogISAPI 66
EDialogSend 67
EDialogSendWinAPI 67
EDialogService 67
EDialogWeb 67
EDialogWER 68
EDialogWinAPI 68
EDialogWinAPIEurekaLog 69
EDialogWinAPIEurekaLogDetailed 69
EDialogWinAPIMSClassic 69
EDialogWinAPIStepsToReproduce 70
EDisAsm 70
EDLLs 71
EDNS 71
EDNSError 130
EEncoding 72
EEncrypt 73
EEurekaLogCurrentVersion 583
EEurekaLogGeneralError 131
EEurekaLogLicense 583
EEurekaLogVersion 584
EEvents 75
EException 81
EExceptionHook 82
EExceptionInfo 82
EExceptionInfoDelphi2 83
EExceptionInfoDelphiANSI 83
EExceptionInfoDelphiUnicode 83
EExceptionInfoGeneric 83
EExceptionInfoMSVC 84
EExceptionManager 84
EFileMemory 85
EFixSafeCallException 85
EFonts 85
EFreeze 86
EHash 87
EHook 89
EHTMLHelp 89
EInfoFormat 89
EInject 90
EInterfaces 90
EJSON 90
EListView 90
ELogBuilder 90
ELogging 92
ELogManager 93
ELowLevel 93
ELowLevelClasses 95
EMantisConnectStub 95
EMAPI 95
EMapParserBase 95
EMapWin32 96
emD2009 535
EMemLeaks 96
emFull 536
emHook 536
EMLang 100
emNull 536
EModules 101
EMonitors 101
emUnknown 536

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

EnableMemLeaksRegistering 294
EncodeConfigValue 295
EncryptBufferFree 295
ENT 101
EOSApiList 101
EPChars 102
EPEImage 102
EPNG 102
EResLeaks 102
ErrorCode 548
ErrorMessage 548
ERunTime 103
EScreenShot 105
ESend 105
ESendAPIBugZilla 106
ESendAPIFogBugz 106
ESendAPIJIRA 107
ESendAPIMantis 107
ESendMail 108
ESendMailMAPI 108
ESendMailShell 109
ESendMailSMAPI 109
ESendMailSMTP 110
ESendWeb 111
ESendWebFTP 111
ESendWebHTTP 111
ESignature 112
ESOAP 113
ESOAPMantis 113
ESocket 113
ESockets 113
ESpecificDelphi 114
ESSLSocket 114
EStackTracing 114
EStrConsts 114
ESysInfo 115
ETaskDialogs 117
EThreadsManager 117
ETools 118
ETypes 118
EUnmangling 119
EurekaAllocMem 297
EurekaCallStackClass 584
EurekaFreeMem 298
EurekaGetMem 298
EurekaReallocMem 299
EurekaRegisterExpectedMemoryLeak 300
EurekaRegisterExpectedMemoryLeakDynArray 301
EurekaRegisterExpectedMemoryLeakStr 302
EurekaTryGetMem 303
EurekaUnregisterExpectedMemoryLeak 303
EurekaUnregisterExpectedMemoryLeakDynArray 304
EurekaUnregisterExpectedMemoryLeakStr 305
EWCT 119
EWCTSupport 120
EWebTools 120
EWER 120
EWinCrypt 121
EWinSock 121
ExceptionClass 167
ExceptionCount 215
ExceptionInfo 137
ExceptionLog7 121
ExceptionManager 305
ExceptionMessage 167
ExceptionNumber 168
ExceptionObject 168
ExceptionRecord 169
Execute 218, 230
EXML 123
EXMLBuilder 123
EXMLRPC 123
EXMLRPCBugZilla 124
ExpandEnvVars 306
ExpectedContext 169
ExpectedURL 170
Extract 240
ExtractAll 241
ExtractZipFile 236
EZip 124
EZIPException 131
EZLib 125
EZLibHeaders 125

- F -

FBuilded 136
FileComment 244
FileCount 244
FileInfo 245
FileInfos 245
FileName 245, 562
FileNames 245
FindDialogClass 307
FindLowAddress 308
FreezeThreadClass 585
ftBugReport 532
ftHTMLPage 532
ftLog 532
ftMiniDump 532
ftSavedBugReport 533
ftScreenshot 533

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

ftXML 533
ftZIP 533
ftZIPCopy 533

- G -

GetActiveFormClass 308
GetActiveFormText 309
GetApplicationDescription 310
GetApplicationName 310
GetApplicationRect 311
GetApplicationWnd 312
GetAsmSize 312
GetCompanyName 313
GetComputerName 314
GetControlClass 314
GetControlText 315
GetCurrentInstruction 150
GetCurrentModuleCompilationDate 316
GetCurrentStackFrame 151
GetCurrentStackPointer 152
GetCurrentTopOfStack 152
GetCurrentUserPrivileges 316
GetDiskInfo 317
GetDisplayDPI 317
GetDisplayMode 318
GetEnvVariable 318
GetEurekaLogLabel 319
GetEurekaLogLicense 320
GetEurekaLogPath 321
GetEurekaLogStatus 322
GetEurekaLogVersion 322
GetExceptionModuleName 323
GetExceptionModuleVersion 324
GetFileIcon 324
GetFileSize 325
GetFunctionSize 326
GetMainWnd 327
GetModuleType 327
GetMXServerFromEmail 328
GetReadableSize 329
GetTaskBarWnd 330
GetTracer 331
GetTracerClass 332
GUIFont 333

- H -

Handle 193, 205
Handled 170
HandleException 333
Handler 171
HookWin32API 334
HostOS 578

- I -

ID 194
IfElse 335
IndexOf 241
Info 171, 208
IniReadString 336
InitCheckFreeze 337
InitializationException 172
InitSalt 338
InitWER 340
IniWriteString 340
InnerException 172
Instance 204
InstructionToString 341
IntoIDE 342
IsAppType 342
IsBPL 343
IsCGI 344
IsCGIConsole 345
IsCGIWin 345
IsCLX 346
IsCompiledWithPackages 347
IsConsole 347
IsCPL 347
IsDebugged 348
IsDLL 349
IsEurekaLogActive 3, 349, 350
IsEurekaLogActiveInThread 5, 352
IsEurekaLogInstalled 2, 353, 354
IsEXE 355
IsFMX 355
IsGUI 356
IsIntraWeb 357
IsISAPI 357
IsMainThread 194
IsMemLeaksEnabled 358
IsMemLeaksInstalled 358
IsModuleLibrary 359
IsReadableFile 360
IsRTLCompiledWithStackFrames 360
IsRTLThread 194
IsService 361
IsStackFramesProc 361
IssueID 549
Issuer 485
IssuerID 486

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

IsValid 237
IsValidBlockAddr 362
IsValidClass 363
IsValidCodeAddr 364
IsValidHandle 365
IsValidObject 365
IsVCL 366
IsWeb 367
IsWin10 368
IsWin64 368
IsWin7 369
IsWin8 369
IsWin81 370
IsWindowsThread 195
IsWinVista 371
IsWinXP 371
IsWorkstation 372
IsWritableFile 373
Items 202

- K -

Kind 195

- L -

LastBugID 9, 373
LastThreadException 215
leHang 542
leInvalidBlock 542
leMultiFree 542
leNone 542
leOutOfMemory 542
leOverflow 543
leUseAfterFree 543
LogBuilderClass 586

- M -

MadeByVersion 578
MainThreadTopOfStack 374
MapStringToFileName 140
MapStringToModuleName 141
MapStringToStr 142
MD5Hash 374
MemLeaksName 375
MemLeaksNameDynArray 376
MemLeaksNameStr 376
MemLeaksOwn 377
MemLeaksOwnDynArrayBoth 378
MemLeaksOwnDynArrayChild 378
MemLeaksOwnDynArrayParent 379
MemLeaksOwnStr 380
MemLeaksSetParentBlock 380
MemLeaksSetParentBlockDynArrayBoth 381
MemLeaksSetParentBlockDynArrayChild 381
MemLeaksSetParentBlockDynArrayParent 382
MemLeaksSetParentBlockStr 382
MemProtect 383
MemUnprotect 384
Mode 216, 245
Module 173
ModuleName 173
ModuleVersion 173
MonoFont 386
MoreInfoLink 562
mtLibrary 544
mtPackage 544
mtProgram 544
mtUnknown 545

- N -

Name 196
NameThread 386
NameThreadForDebugging 387
Next 538
NotAfter 486
NotBefore 487
Notify 209

- O -

OnAttachedFilesRequest 182
OnBeginReportGen 182
OnCustomBugID 183
OnCustomButtonClick 183
OnCustomDataRequest 183
OnCustomFileName 184
OnCustomWebFieldsRequest 184
OnEndReportGen 184
OnExceptionAction 185
OnExceptionError 185
OnExceptionNotify 186
OnPasswordRequest 186
OnRaise 186
OnReproduceRequest 187
OnZippedFilesRequest 187
Open 242
Options 138, 174

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

- P -

PageSize 388
Parameter 196
ParentID 197
PauseFreezeCheck 388
PCustomName 480
PInstruction 480
PlatformSpecific 538
PPointerRecord 480
PrepareStr 143
PrimaryInstance 389
Priority 191
PrivateKey 553
Processed 174
ProgramName 563
PSalt 481
PTwofishInitVector 481
PublicKey 553
PublisherLink 563

- R -

RaiseExpected 389
RaiseFreezeException 390
Read 242
Reader 175
RealParameter 197
RealStartingAddress 197
RealStartingFunction 198
Refresh 134
RegisterDialogClass 391
RegisterDialogClassFirst 391
RegisterEventAttachedFilesRequest 392
RegisterEventBeginReportGen 393
RegisterEventCustomBugID 394
RegisterEventCustomButtonClick 395
RegisterEventCustomDataRequest 396
RegisterEventCustomFileName 397
RegisterEventCustomWebFieldsRequest 398
RegisterEventEndReportGen 399
RegisterEventExceptionAction 400
RegisterEventExceptionError 401
RegisterEventExceptionNotify 402
RegisterEventHelpButtonClick 403
RegisterEventPasswordRequest 404
RegisterEventRaise 405
RegisterEventReproduceRequest 406
RegisterEventZippedFilesRequest 407
RegKeyCheck 408
RegKeyRead 408
RegKeyWrite 409
Report 138
ReproduceText 175
RequiredVersion 579
ResumeFreezeCheck 410
RSADecrypt 411
RSADestroyKey 414
RSAEncrypt 414
RSAGenKey 417
RSALoadPrivateKey 418
RSALoadPublicKey 419
RSASavePrivateKey 420
RSASavePublicKey 421
RSASign 422
RSAVerify 424
rsBLOB 551
rsDER 552
rsPEM 552

- S -

SafeCallExpt 176
SafeExec 426
SDBMHash 427
SecureFree 428
SendResult 549
Serial 487
SetAppType 429
SetEurekaLogState 6, 430, 431
SetEurekaLogStateInThread 7, 432
SetupConfig 433
SHA1Hash 434
SHA256Hash 434
ShowExpected 435
ShowLastExceptionData 211
SIGNATURE_CENTRALHEADER 586
SIGNATURE_DATADESCRIPTOR 587
SIGNATURE_LOCALHEADER 587
SIGNATURE_ZIPENDOFHEADER 588
SignatureGetInfo 436
Signed 564
SingleSize 547
srBugClosed 557
srCancelled 557
srConnectionError 557
srInvalidInsert 558
srInvalidLogin 558
srInvalidModify 558
srInvalidRead 558
srInvalidSearch 558

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

srInvalidSelection 559
srNoExceptionInfo 559
srObsoleteVersion 559
srRestart 559
srSent 559
srUnknownError 560
srUnsupportedSender 560
StackDump 176
StackPoint 177
StackTop 177, 198
StandardEurekaError 212
StandardEurekaNotify 213
Start 539
StartingAddress 199
StartingFunction 199
Subject 488
SubjectID 488
SuccessMessage 550
SynchronizeException 178

- T -

TAppType 482
TargetThreadHandle 219
TargetThreadID 219
TargetWindow 219
TBaseDialog 32, 131
TBaseDialogClass 482
TBaseLogBuilder 132
TBlockType 483
TBugID 484
TCertificateInfo 485
TCheckType 489
TCRC16Hash 491
TCRC32Hash 492
TCustomName 492
TDummyLogBuilder 139
TDumpCompareFunc 493
TDumpFilterFunc 493
TEADecrypt 437
TEADefaultKey 441
TEADeriveKey 443
TEADestroyKey 447
TEAEncrypt 447
TEAInitSessionKey 451
TELAbstractMapParser 139
TELEvAttachedFilesRequestMeth 19, 494
TELEvAttachedFilesRequestProc 495
TELEvCustomBugIDMeth 22, 496
TELEvCustomBugIDProc 496
TELEvCustomButtonClickMeth 24, 497
TELEvCustomButtonClickProc 498
TELEvCustomDataRequestMeth 23, 499
TELEvCustomDataRequestProc 500
TELEvCustomFileNameMeth 21, 501
TELEvCustomFileNameProc 501
TELEvCustomWebFieldsRequestMeth 23, 502
TELEvCustomWebFieldsRequestProc 503
TELEvExceptionActionMeth 15, 504
TELEvExceptionActionProc 505
TELEvExceptionErrorMeth 17, 507
TELEvExceptionErrorProc 508
TELEvExceptionNotifyMeth 14, 510
TELEvExceptionNotifyProc 511
TELEvHelpButtonClickMeth 25, 513
TELEvHelpButtonClickProc 514
TELEvPasswordRequestMeth 18, 515
TELEvPasswordRequestProc 516
TELEvRaiseMeth 12, 517
TELEvRaiseProc 518
TELEvReportGenMeth 26, 520, 521
TELEvReportGenProc 521, 522
TELEvReproduceRequestMeth 21, 523
TELEvReproduceRequestProc 523
TELEvZippedFilesRequestMeth 20, 524
TELEvZippedFilesRequestProc 525
TELFTPSender 39, 144
TELHTTPSender 39, 145
TELMailMAPISender 38, 145
TELMailShellSender 37, 145
TELMailSimpleMAPISender 37, 146
TELMailSMTPBaseSender 146
TELMailSMTPClientSender 39, 146
TELMailSMTPSender 38, 147
TELMailSMTPServerSender 38, 147
TELTrackerBugZillaSender 147
TELTrackerFogBugzSender 41, 148
TELTrackerJIRASender 148
TELTrackerMantisSender 40, 149
TEncryptBuffer 525
TEurekaActionType 526
TEurekaBaseStackList 149
TEurekaCallStack 530
TEurekaCallStackClass 531
TEurekaExceptionInfo 157
TEurekaLogEvents 29, 180
TEurekaLogFileType 531
TEurekaModuleOptions 188
TEurekaModuleType 533
TEurekaStackList 534
TEurekaStackListV6 188
TEurekaStackListV7 189
TEurekaThreadData 190

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

TEurekaThreadDataList 200
TExceptionManager 203
TExceptionNotifyMode 534
TFreezeThread 217
TFreezeThreadClass 536
TGenericStackList 220
Thread 200
ThreadClassName 200
ThreadData 202, 452
ThreadDataEnum 453
ThreadException 178
ThreadID 179
Time 179
Timestamp 565
TimestampCertificate 565
Timestamped 566
TInstruction 537
tkBeginThread 568
tkGeneric 568
tkTThread 569
TLogBuilder 223
TMainThreadFreezeDetectionThread 224
TMD5Hash 540
TMemoryError 540
TMemoryErrors 543
TModuleType 543
ToString 135, 157
TPointerRecord 545
Tracer 222
TResponse 547
TRSAExport 551
TRSAKey 552
TRSASignature 554
TSalt 554
TSDBMHash 555
TSendResult 555
TSHA1Hash 560
TSHA256Hash 560
TSignatureInfo 561
TStreamConstructor 567
TTEAKey 567
TThreadEx 225
TThreadKind 568
TTwofishInitVector 569
TTwofishKey 569
TTwofishRAWKey 570
TWCTFreezeDetectionThread 234
TwofishDecrypt 454
TwofishDeriveKey 455
TwofishDestroyKey 457
TwofishEncrypt 457
TwofishInitIV 459
TwofishInitKey 459
TXMLLogBuilder 235
TZipCompression 570
TZipCompressionToString 460
TZipEndOfCentralHeader 576
TZipFile 235
TZipHeader 577

- U -

UnhandledExceptionCount 216
UnhandledExceptionNumber 179
UnRegisterEventAttachedFilesRequest 461
UnRegisterEventBeginReportGen 462
UnRegisterEventCustomBugID 462
UnRegisterEventCustomButtonClick 463
UnRegisterEventCustomDataRequest 464
UnRegisterEventCustomFileName 465
UnRegisterEventCustomWebFieldsRequest 466
UnRegisterEventEndReportGen 466
UnRegisterEventExceptionAction 467
UnRegisterEventExceptionError 468
UnRegisterEventExceptionNotify 469
UnRegisterEventHelpButtonClick 470
UnRegisterEventPasswordRequest 470
UnRegisterEventRaise 471
UnRegisterEventReproduceRequest 472
UnRegisterEventZippedFilesRequest 473

- V -

Valid 539
Version 489

- W -

WCTDone 474
WCTInit 474

- Z -

zcBZIP2 572
zcDeflate 572
zcDeflate64 572
zcImplode 573
zcLZ77 573
zcLZMA 573
zcPKImplode 573
zcPPMdI1 573
zcReduce1 574

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.

zcReduce2 574
zcReduce3 574
zcReduce4 574
zcReserved11 574
zcReserved13 575
zcReserved15 575
zcReserved16 575
zcReserved17 575
zcShrunk 575
zcStored 576
zcTERSE 576
zcTokenize 576
zcWavePack 576
ZipDirectoryContents 237

© 2001-2015, Neos Eureka S.r.l. All Rights Reserved.