EurekaLog 7.x.x user manual

Table of Contents

Foreword 0

Part I Welcome to EurekaLog 7 2

Part II What's New in EurekaLog 7.0 4

Part III Introduction 8

1 Features / Editions ... 10

2 How to buy ... 14

3 Support ... 16

Part IV Quick start tutorials 20

1 Installation ... 20

Using EurekaLog with Delphi Standard/Personal/Turbo/Starter editions ... 23

Installation for non-admin user account ... 31

Installation for AppWave or without IDE installed ... 31

2 How to use EurekaLog ... 33

3 Video Tutorials ... 0

4 EurekaLog's basics ... 38

5 Basic terms (definitions/dictionary) ... 40

Part V Basic procedures 45

1 Video Tutorials ... 0

2 Selecting application type ... 45

3 Configuring bug report ... 46

Configuring call stack ... 48

4 Configuring dialogs ... 52

5 Configuring sending report ... 53

Selecting send method ... 55

6 Configuring project itself ... 58

Part VI Typical scenarios 68

1 Reporting ... 68

2 Debugging ... 70

Part VII Solving bugs in your code 72

1 Bug reports ... 72

General section ... 74

Call Stack section ... 78

Call stacks ... 79

How to read call stacks ... 81

Call stack formats ... 83

Multi-threaded call stacks ... 85

Searching bug’s location ... 95

Final notes on call stack ... 96

Searching bug's reason ... 97
Modules section ... 99
Processes ... 99
Assembler ... 100
CPU ... 103
Screenshot ... 105
Additional files ... 105
2 Managing bug reports in issue tracker ... 105
Bug trackers setup ... 107
FogBugz setup ... 108
Using HTTP upload ... 115
Mantis setup ... 119
BugZilla setup ... 134
JIRA setup ... 143
Using unsupported bug tracker software ... 153
Customizing feedback ... 155
Security Considerations ... 158
3 EAccessViolation ... 160
4 Leaks ... 166
5 Memory problems ... 171
6 Hangs and deadlocks ... 174
Part VIII Customizing EurekaLog 180
1 EurekaLog options ... 180
Static options ... 180
Dynamic options ... 183
Variables ... 183
Filters ... 185
2 Coding ... 189
Changing options at run-time ... 189
Custom attributes ... 190
Events ... 192
Subclassing ... 195
Low-level handlers ... 211
Modifying code of EurekaLog itself ... 212
Part IX Frequently Asked Questions (FAQ) 215
1 General FAQ ... 215
2 Default files names and locations ... 217
3 File formats ... 218
Part X Integral parts 221
1 EurekaLog IDE expert ... 221
2 Interface ... 222
IDE menu items (IDE commands) ... 222
Project options ... 225
Import / export settings ... 227
Using variables ... 228
Changing default properties ... 230
IDE options ... 230
3 Options ... 234
General page ... 234

Features page **237**
Call Stack page 237
BugID page 240
Debug information page 243
Nested exceptions page 244
Multi-threading page 246
Memory problems page 250
Enabling memory/resource leaks features for C++ Builder 255
Resource leaks page 255
Hang detection page 257
Restart&Recovery page 259
External tools 259
Bug report page **264**
Bug report content page 266
Dialogs page **267**
MessageBox 268
MS Classic 271
EurekaLog 279
Console 292
System log reporting 295
WEB 296
Windows Error Reporting 300
Report sending page **302**
Sending options page 304
Shell send 309
Simple MAPI 314
MAPI 315
SMTP client 316
SMTP server 319
HTTP upload 320
FTP upload 321
FogBugz 322
Mantis 327
BugZilla 331
JIRA 335
Localization page **339**
Advanced page **341**
Exceptions filters page 343
Editing exception filter 344
Build options page 349
Build events page 351
Code page 352
Hooks page 352
Dialogs page 354
Debug information page 355
Send engines page 355
Custom/Manual page 356
3rd party page **357**
Statistics **357**
4 Application types **363**
VCL Forms Application **365**
CLX Forms Application **366**
FireMonkey application **366**
VCL Control Panel Application **366**
VCL NT Service Application **367**
VCL CGI Application **367**
VCL ISAPI Application **367**
VCL IntraWeb Application **368**

Console Application ... 368
DLL ... 368
DLL (standalone) ... 369
Package ... 370
Unsupported application types ... 370
5 Dialogs ... 370
(none) ... 371
RTL ... 371
Message box ... 373
Windows Classic ... 377
EurekaLog ... 379
Console ... 382
System log ... 384
WEB ... 386
WER ... 389
6 Send engines ... 390
Shell ... 391
Simple MAPI ... 393
MAPI ... 396
SMTP client ... 397
SMTP server ... 398
HTTP upload ... 398
FTP upload ... 404
FogBugz ... 404
Mantis ... 406
BugZilla ... 407
JIRA ... 408
7 Debug information providers ... 409
EurekaLog ... 410
.map file ... 410
TD32 ... 411
Exports table ... 411
JCL ... 412
Dbg/Pdb ... 412
madExcept ... 413
8 Variables ... 413
Part XI Advanced topics 421
1 BugID ... 421
2 Compiling your project with EurekaLog ... 421
EurekaLog post-process compilers ... 423
Minimum parameters needed ... 424
Post-processing without (re)compilation ... 426
Using EurekaLog with automated builds ... 429
Delphi 2007+ ... 429
FinalBuilder ... 431
ecc32/emake command line options ... 432
3 Working with configurations ... 439
Storing EurekaLog options ... 440
EurekaLog options syntax ... 442
Using external configuration ... 443
Compiling your project with and without EurekaLog ... 445
Different EurekaLog settings for 'Debug' and 'Release' profiles ... 448
Read-only projects ... 449
Sharing EurekaLog settings in project group ... 450

Reconfiguring EurekaLog for manual control ... 451

4 Using EurekaLog in DLL ... 455

Introduction ... 455

What is DLL ... 455

What is exception ... 456

How DLLs report about failures ... 457

What is the proper way to handle exceptions in DLL ... 457

Framework ... 458

System or 3rd party API ... 459

Your own API ... 467

Creating bug reports for DLL exceptions ... 470

Using exception tracer tool in DLLs ... 473

Single instance of exception tracer ... 474

Multiple instances of exception tracer ... 480

Using exception tracer with frameworks in DLLs ... 484

Single instance of exception tracer ... 484

Multiple instances of exception tracer ... 487

Using exception tracer with COM objects ... 488

Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) ... 495

Using EurekaLog with non-Embarcadero DLLs ... 496

Using Microsoft DbgHelp DLL ... 504

5 Configuring project for leaks detection ... 508

About leak detection ... 508

Configurations ... 512

Typical application ... 512

DLLs ... 513

Shared memory manager ... 513

Packaged project ... 514

6 Using EurekaLog with other software ... 514

Debuggers and profilers ... 515

Debug information converters ... 516

Digital signature tools ... 519

EXE packers, EXE cryptors, EXE protectors ... 520

Localization software ... 524

Shared memory manager ... 524

Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) ... 526

Using EurekaLog with non-Embarcadero DLLs ... 527

7 System logging setup ... 535

System Log ... 536

Registering Event Source ... 537

Configuring dialog ... 543

Using dynamic content with System Log ... 543

8 Multi-threaded applications ... 547

Creating threads ... 547

CreateThread Function ... 548

BeginThread Function ... 549

BeginThreadEx Function ... 551

TThread Class ... 553

TThreadEx class ... 559

Anonymous threads ... 562

Frameworks ... 565

Thread Pools ... 566

Enabling EurekaLog for background threads ... 568

Automatic / options ... 569

Manual / code ... 570

9 Using Windows Error Reporting ... 573

10 Working with minidumps 573
11 Nested/Chained exceptions 573
12 Stack tracing: RAW method and frame-based method 578
13 Stack tracing: deferred vs. immediate 583
14 EurekaLog for shareware developers 585
15 What's the difference between SSL and TLS send modes? 588
16 Memory leaks detection limitations 589
17 Resource leaks detection limitations 589
List of functions 590
18 Internal Errors 591

Part XII Troubleshooting 596

1 Troubleshooter 0
2 Installation problems 596
Manual installation 598
Where to find EurekaLog 603
EurekaLog IDE Setup 604
IDE names mapping 604
3 Uninstallation problems 605
4 Enabling EurekaLog problems 606
Verifying that EurekaLog was enabled 610
5 EurekaLog run-time problems 613
6 Breakpoints 614
7 Knowledge base 0
8 Other problems 615

Part XIII Tools 617

Part XIV Recompilation (Enterprise) 619

1 Files layout 619
2 Recompilation 620

Part XV Compatibility 624

1 6.x -> 7.x 624
Upgrage guide 627
2 5.x -> 6.x 633
What's New in EurekaLog 6.0 634
What's New in EurekaLog 6.1 634
3 4.5.x -> 5.x 636
4 4.x -> 4.5.x 637
5 3.x -> 4.x 637

Part XVI License 639

Index 641

Part I

1 Welcome to EurekaLog 7

EurekaLog helps you to **find and resolve bugs** in your code, including exceptions, leaks and hangs.

EurekaLog is the **exception tracer tool** (what's this? 40) that gives your application the ability to catch all bugs in your code and generates a detailed log with the call stack, which includes unit, class, method and line number information (example 8). This information is also logged to a file and may optionally be forwarded to you (application developer) via Internet.

Getting started

New users

- View Introduction 8 or Screenshots
- Study Quick Start Tutorials 20 or Video Tutorials to familiarize yourself with the very basics of the EurekaLog
- You can read Basic terms 40 if you're new to exception handling
- You can read Typical scenarios 68 to get idea on how to use EurekaLog to troubleshot problems
- You can study manuals on basic procedures 45 to continue learn about EurekaLog
- There are plenty of help and demos available! You can find them in EurekaLog's Start Menu folder. Also don't forget to check out our FAQ 215 section
- In case of any problems - see our Troubleshooting 596 section

Users upgrading from previous version

- Study changes 4 in EurekaLog
- Even if you are an experienced EurekaLog user, please run through the Introduction 8 and Quick Start Tutorials 20 sections quickly to get up to speed with what has changed in the latest version of the EurekaLog
- **Study migration guide 627, migration reference 624 and typical scenarios 68**
- You can read description of EurekaLog's integral parts 221 or study code's reference

Part II

2 What's New in EurekaLog 7.0

EurekaLog 7 includes new features, enhancements and changes in the following areas:

(see most recent changelog here)

EurekaLog

(in random order)

- **Improved:** Main change - EurekaLog's core was rewritten (refactored) to allow more easy modification and remove hacks.
- **Improved:** New plugin-like architecture now allows you to exclude unused code.
- **Improved:** New plugin-like architecture now allows you to easily extends EurekaLog.
- **Improved:** Greatly extended documentation.
- **Improved:** Installer is now localized.
- **Improved:** Greatly speed ups creation of minimal bug report (with most information disabled).
- **Changed:** EurekaLog's root IDE menu was relocated to under Tools and extended with new items.
- **Added:** New examples.
- **Added:** New tools 617.
- **Added:** Support for DBG/PDB formats of debug information (including symbol server support and auto-downloading).
- **Added:** Support for madExcept debug information (experimental).
- **Added:** WER (Windows Error Reporting) support.
- **Added:** Full unicode support.
- **Added:** Professional and Trial editions: added source code (interface sections only)
- **Improved:** Dialogs - new options and new customization possibilities:
  - **Added:** All GUI dialogs: ability to test dialog directly from configuration dialog by displaying a sample window with currently specified settings.
  - **Improved:** All GUI dialogs: dialogs are DPI-awared now (auto-scale for different DPI).
  - **Added:** MessageBox dialog: added detailed mode (shows a compact call stack).
  - **Added:** MessageBox dialog: added ability for asking a send consent.
  - **Added:** MessageBox dialog: added support to switch to "native" message box for application.
  - **Added:** MS Classic dialog: added control over "user e-mail" edit's visibility.
  - **Added:** MS Classic dialog: added ability to personalize dialog view with application's name and icon.
  - **Added:** MS Classic dialog: added ability to show terminate/restart checkbox initially checked.
  - **Added:** EurekaLog dialog: added ability to personalize dialog view with application's name and icon.
  - **Added:** EurekaLog dialog: added ability to show terminate/restart checkbox initially checked.
  - **Added:** EurekaLog dialog: added ability to switch back to non-detailed view.
  - **Added:** WEB dialog: added new tags to customize bug report page.
  - **Improved:** WEB dialog: improved support for unicode and charset.
  - **Added:** New dialog type: RTL dialog.
  - **Added:** New dialog type: console output.
  - **Added:** New dialog type: system logging.
  - **Added:** New dialog type: Windows Error Reporting.
- **Improved:** Sending - new options and new customization possibilities:
  - **Added:** All send methods: added ability to setup multiple send methods.
  - **Added:** All send methods: added ability to change send method order.
  - **Added:** All send methods: added separate settings for each send method.
  - **Added:** All send methods: ability to test send method directly from configuration dialog by sending a demo bug report.
  - **Added:** SMTP client send method: added SSL support.
  - **Added:** SMTP client send method: added TLS support.
  - **Added:** SMTP client send method: added option for using real e-mail address.

  - **Added:** SMTP server send method: added option for using real e-mail address.
  - **Added:** HTTP upload send method: added support for custom backward feedback messages.
  - **Added:** FTP upload send method: added creating folders on FTP (like remote ForceDirectories).
  - **Added:** Mantis send method: added API support (MantisConnect, out-of-the-box since Mantis 1.1.0, available as add-on for previous versions).
  - **Added:** Mantis send method: added support for custom "Count" field.
  - **Added:** Mantis send method: added options for controlling duplicates.
  - **Added:** Mantis send method: added support for SSL/TLS.
  - **Added:** FogBugz send method: added API support (out-of-the-box since ForBugz 7, available as add-on for FogBugz 6).
  - **Added:** FogBugz send method: EurekaLog will update "Occurrences" field (count of bugs).
  - **Added:** FogBugz send method: EurekaLog will respect "Stop reporting" option (BugzScout's setting).
  - **Added:** FogBugz send method: EurekaLog will respect "Scout message" option (BugzScout's setting).
  - **Added:** FogBugz send method: EurekaLog will store client's e-mail as issue's correspondent.
  - **Added:** FogBugz send method: added options for controlling duplicates.
  - **Added:** FogBugz send method: added support for "Area" field.
  - **Added:** FogBugz send method: added support for SSL/TLS.
  - **Added:** BugZilla send method: added API support.
  - **Added:** BugZilla send method: added support for custom "Count" field.
  - **Added:** BugZilla send method: added options for controlling duplicates.
  - **Added:** BugZilla send method: added support for SSL/TLS.
  - **Added:** New send method: Shell (mailto protocol).
  - **Added:** New send method: extended MAPI.
- **Added:** Support for separate code and debug info injection.
- **Added:** Ability to use custom units before EurekaLog's units.
- **Added:** Support for external configuration file in IDE expert.
- **Added:** Now EurekaLog stores only those project options which are different from defaults (to save disk space and reduce noise in project file).
- **Added:** Now EurekaLog stores project options sorted (alphabet order).
- **Added:** Separate settings for saving modules and processes lists to bug report.
- **Added:** Support for taking screenshots of multiple monitors.
- **Added:** More screenshot customization options.
- **Added:** More control over bug report's file names.
- **Added:** New environment variables.
- **Added:** Deleting .map file after compilation.
- **Added:** Support for different .dpr and .dproj file names.
- **Improved:** memory leaks detection feature - new options and new customization possibilities:
  - **Added:** Ability to track memory problems without activation of leaks checking.
  - **Added:** Support for sharing memory manager.
  - **Added:** Support for tracking leaks in applications built with run-time packages.
  - **Added:** Option to zero-fill freed memory.
  - **Added:** Option to enable leaks detection only when running under debugger.
  - **Added:** Option for manual activation control for leaks detection (via command-line switches).
  - **Added:** Option to select stack tracing method for memory problems.
  - **Added:** Option to trigger memory leak reporting only for large leaked memory's size.
  - **Added:** Option to control limit of number of reported leak.
  - **Added:** `CheckHeap` function to force check of heap's consistency.
  - **Added:** `DumpAllocationsToFile` function to save information about allocated memory to log file.
  - **Added:** Registered leaks feature.
  - **Added:** Run-time control over memory leak registering.
  - **Added:** New recognized leak type: String (both ANSI and Unicode are supported).
  - **Added:** Memory features support for C++ Builder.
- **Added:** Resource leaks detection feature.

- **Improved:** Compilation speed increased.
- **Added:** Support for generics in debug information.
- **Added:** Chained/nested exceptions support.
- **Added:** Wait Chain Traversal support.
- **Added:** Support for named threads.
- **Added:** Additional information for threads in call stack.

EurekaLog Viewer

- See What's new in EurekaLog Viewer section!

Compatibility issues

- See the Changes from 6.x version 624 section!

Features list

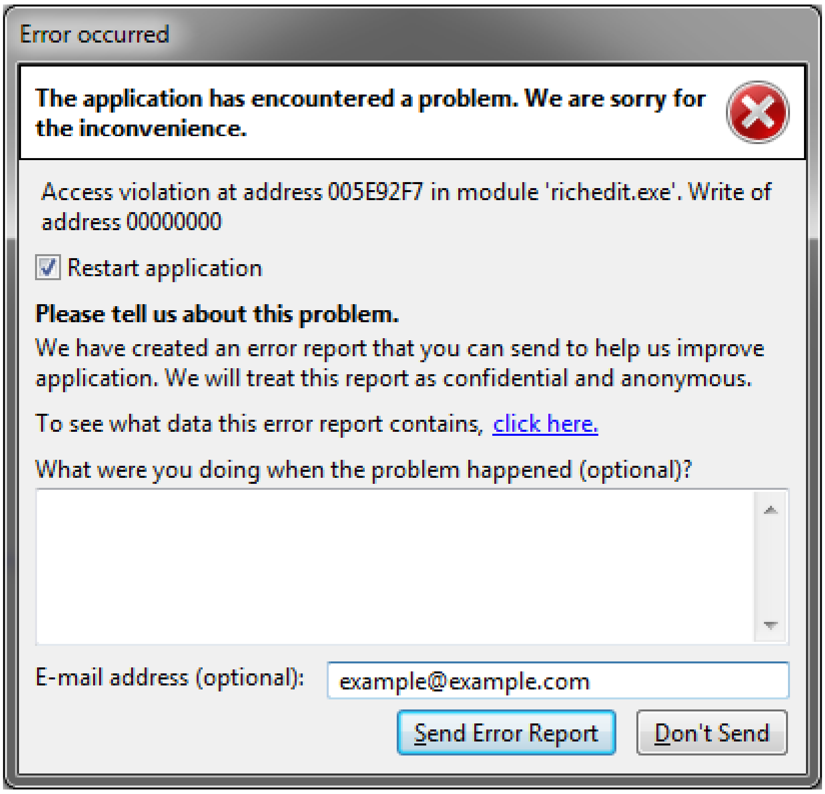
- See the "Features" 10 page!

Part III

3 Introduction

EurekaLog helps you to **find and resolve bugs** in your code, including exceptions, leaks and hangs.

EurekaLog is the exception tracer tool that gives your application (GUI, Console, Web, etc.) the ability to catch all bugs in your code and generates a detailed log with the call stack, which includes unit, class, method and line number information as shown in the image below. This information shown is also logged to a file and may optionally be forwarded to you via Internet.



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E92F7 in module 'richedit.exe'. Write of address 00000000

☑ Restart application

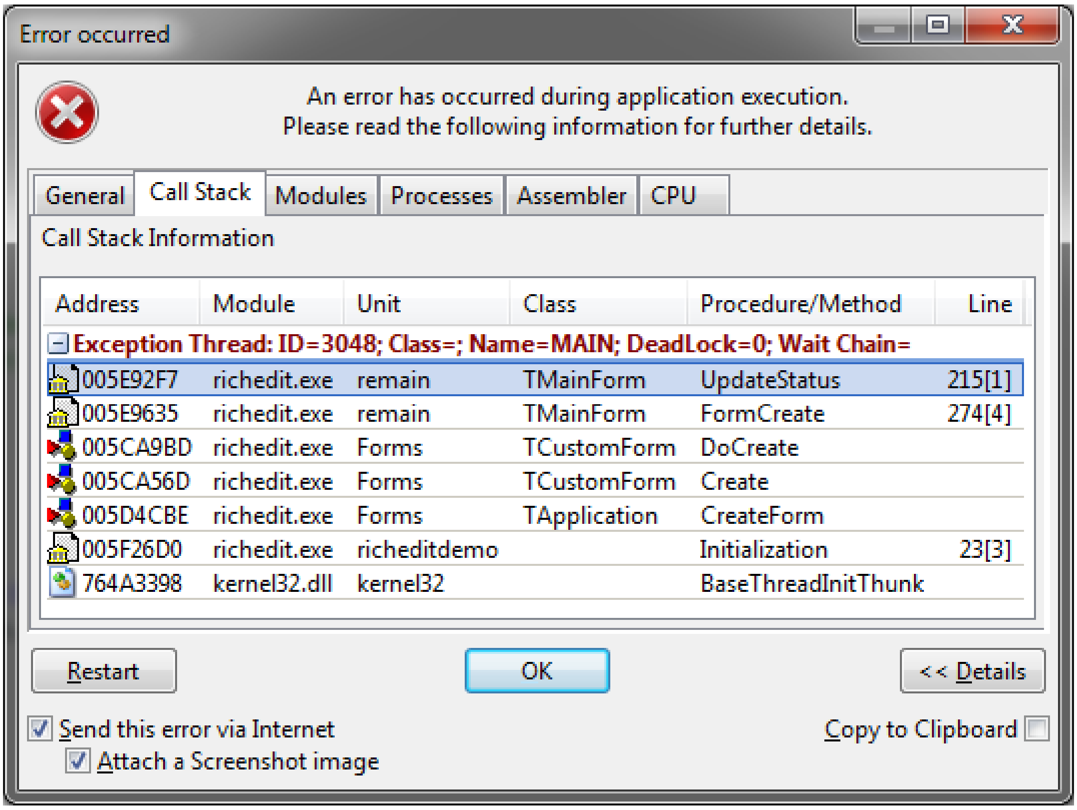
Please tell us about this problem.

We have created an error report that you can send to help us improve application. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

What were you doing when the problem happened (optional)?

E-mail address (optional): example@example.com



Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=3048; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
005E92F7	richedit.exe	remain	TMainForm	UpdateStatus	215[1]
005E9635	richedit.exe	remain	TMainForm	FormCreate	274[4]
005CA9BD	richedit.exe	Forms	TCustomForm	DoCreate	
005CA56D	richedit.exe	Forms	TCustomForm	Create	
005D4CBE	richedit.exe	Forms	TApplication	CreateForm	
005F26D0	richedit.exe	richeditdemo		Initialization	23[3]
764A3398	kernel32.dll	kernel32		BaseThreadInitThunk	

☑ Send this error via Internet
☑ Attach a Screenshot image
Copy to Clipboard □

Examples of typical EurekaLog dialogs

EurekaLog is easy to use because it is fully integrated into the Delphi/C++Builder IDE. You can just enable EurekaLog for your application and rebuild your application - that's it: now you have all additional powers of EurekaLog!

EurekaLog does not affect the performance of your application, as it only executes when an exception is raised. It increases the compiled file size by about 0.5% - 4% (this space is needed to store some additional, compressed and encoded, debugging information). EurekaLog doesn't require any additional files to work. You have to distribute only your single executable.

EurekaLog is compatible with Delphi 3, 4, 5, 6, 7, 2005, 2006, 2007, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (*), and with C++Builder 5, 6, 2006, 2007, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (**). It works on Windows platforms, from Windows 95 to Windows 10 (***) - both for Win32 and Win64, including support for Wine, ReactOS and virtual machines. It supports any type of application (****).

It comes with full source (only Enterprise version), full money back guarantee, is royalty free, and freely updatable!

Don't waste anymore time and money debugging your applications: now EurekaLog debugs them for you.

See Features / Editions 10 topic for further details.
See Installation 20 topic for installation process.
See How to use EurekaLog 33 for enabling EurekaLog in your applications.

(*) Delphi 1 and 2 are not supported. Personal, Turbo and Starters editions are supported 23. Delphi 3 is supported by EurekaLog 4, 5 and 6 only.
(**) C++ Builder 1, 3 and 4 are not supported. C++ Builder 5 is supported by EurekaLog 6 only.
(***) Windows 95, 98, ME and NT are supported by EurekaLog 4, 5 and 6 only.
(****) EurekaLog comes with predefined templates "out-of-the-box" for common application's types. Some application types doesn't have predefined settings and require you to setup them manually or

to write additional code.

3.1 Features / Editions

EurekaLog contains all the features that you need in a bug resolution system.

It comes in 3 editions:

- **Trial** - it's fully functional edition with (*one* and *only*) additional limitation: any application, which is compiled with Trial edition of EurekaLog will expire after 30 days. There will be error message box after 30 days and application will exit. Trial itself may be used for infinite time. This edition can be used to evaluate EurekaLog. You can not use this edition for any commercial development.
- **Professional** - it's fully function edition. This is minimum edition to do actual work except evaluating (including commercial development).
- **Enterprise** - it's the same as Professional edition, except it additionally offers full source code of EurekaLog.

Please note that we also have different license types:

- **Single** license - single developer.
- **Company** license - unlimited company developers at one geographical address.
- **Corporate** license - unlimited company developers at unlimited geographical addresses.

EurekaLog also grants you access to all previous EurekaLog versions (4, 5 and 6).

See How to buy[14] topic for more information about license's differences.

If you have any questions - just ask us[16]!

See the following table for edition's differences.

Edition's differences	**Trial**	**Pro**	**Ent**
Expiration			
Compiled application will expire and refuse to run after 30 days since its compilation	✓	✗	✗
Source code			
Source code's interface sections ("headers")	✓	✓	✗
Full source code (*)	✗	✗	✓
(*) While we offer full source code for EurekaLog itself, source code or scripts for external tools may be not available in general installation. This includes setup/build scripts and digital certificate. Please also note that some external tools (like EurekaLog Viewer) requires DevExpress suite for recompilation. You don't need DevExpress to use EurekaLog, but you'll need it if you want to recompile some EurekaLog tools by yourself. EurekaLog itself can be recompiled on any IDE without any additional requirements.			
Common features	**Trial**	**Pro**	**Ent**
Supported languages & Operating Systems			
Delphi versions **3-7, 2005-2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin** (*)	✓		
C++Builder versions **5-6, 2006-2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin** (**)	✓		
Windows **95,98,ME,NT,2000/2003/2008/2008 R2/2012/2012 R2/XP/Vista/7/8/8.1/10, ReactOS, Wine** (***)	✓		
Application types: **ANY** (****)	✓		
(*) Delphi 1 and 2 are not supported. Personal, Turbo and Starter editions are supported[23]. Delphi			

3 is supported by EurekaLog 4, 5 and 6 only.
(**) C++ Builder 1, 3 and 4 are not supported. C++ Builder 5 is supported by EurekaLog 4, 5 and 6 only.
(***) Windows 95, 98, ME and NT are supported by EurekaLog 4, 5 and 6 only.
(****) Some application types may require additional actions.

Common features

EurekaLog contains all the features that you need in a bug resolution system:

- **Detailed bug report about each exception, leak or hang;**
- **Bug report includes call stack with unit names, class names, routine names, and line numbers;**
- **Extensive run-time and environment information is logged into bug report;**
- **RAW dump and disassembly information;**
- **Easy integration, no need to write code;**
- **No additional files needed (no DLLs, no .map files, no .tds files);**
- **Packing and encryption of all information;**
- **No performance loss (unless exception occurs);**
- **Full unicode support;**
- **Win32 and Win64 support;**
- **VCL, CLX, FMX (FireMonkey) support;**
- **Supports any application kind: GUI, CGI, WinCGI, ISAPI, IntraWeb, COM, Multi-Thread, etc.;**
- **Full support for .exe packers and protectors;**
- **Support for modern cutting-edge features (nested exceptiond, Wait Chain Traversal, etc.);**
- **Easy and powerfull customization;**
- **Many helper tools;**
- **Sending bug report to developers (e-mail, HTTP, FTP, bug trackers);**
- **SSL/TLS support for all send methods;**
- **Support for Mantis, FogBugz, BugZilla; More to come...**

Full features list

Common info

- Delphi versions: 4-7, 2005-2010, XE-XE8, 10 Seattle, 10.1 Berlin (including Personal, Turbo and Starter editions)
- C++Builder versions: 6, 2006-2010, XE-XE8, 10 Seattle, 10.1 Berlin (including Turbo and Starter editions)
- Application type: any
- Windows versions: 2000/2003/2008/2008 R2/2012/2012 R2/XP/Vista/7/8/8.1/10
- Other OS: ReactOS, Wine
- Frameworks: VCL/CLX/FMX (FireMonkey)
- Code size: 400-800 Kb (depends on your settings)
- Data size: 1%-12% (depends on your settings)
- Performance decrease: 0%-5% (depends on your settings)
- Unicode support
- ZLib compression
- TEA 128-bit data encryption
- Full and easy customization
- Additional useful tools
- Extensive documentation
- Lots of demos

IDE

- Full integration with Delphi/C++Builder/RAD Studio IDEs
- F1 context help
- Opening source file and positioning text cursor to error line - by double-clicking in error dialogs

- Test both dialogs and sending right in project options dialog
- Support for searching error location even after source file modifications
- Support for __history folder (can show older copy of source)
- Tools integration in IDE
- Revisited IDE menu, options and dialogs
- Pre- and post-build events (can run custom applications)
- Command-line compilation support
- Unicode support

Documentation
- CHM Help file
- Printable PDF manual
- On-line interactive documentation

Features
- Catch any exception (unhandled, handled, safecall, tread, initialization/finalization)
- Catch any memory leak
- Catch any resource leak
- Catch any hang or deadlock
- Nested exceptions support
- Wait Chain Traversal support
- Multi-threading features
- Track exception duplicates via BugID value
- Can sort error by "popularity" ("count" field in web trackers)
- Customizable error dialogs and error web-pages
- Exception filters allow customizations without writing code
- Environment variables (and pseudo-variables) can be used to create run-time dependent options
- Options can be customized in run-time
- Events can be used for arbitrary customizations
- Custom classes can be used for 3rd party extensions
- Restart&Recovery options

Applications
- VCL Forms
- DLLs (both standalone and integrated with EurekaLog-enabled host application)
- BPLs (packages)
- Console
- Control panel applets
- Win32 Services
- ISAPI
- CGI
- WinCGI
- IntraWeb
- ActiveX
- COM-applications
- Multi-threaded
- Indy
- Applications compiled with run-time packages
- Support for .exe compressors
- Support for .exe protectors
- ANY other application kind!

Bug reports
- Bug report collects information about exception, application and run-time environment
- Bug report can be saved to disk, displayed in dialog or sent to developer
- Common information includes info about application, faulted module, system, hardware, user, etc.
- Call stacks of any thread
- Loaded modules list
- Running processes list
- Code disassembly

- CPU state
- Memory and stack dumps
- Plain-text, packed or XML formats
- Can include screenshots (PNG), last web-page (for web apps.) or arbitrary files
- Can include custom data (provided by your code)
- Can be packed (ZIP) and encrypted (TEA); Encrypted reports can be decrypted by Viewer tool
- Unicode support
- Full customization

Dialogs
- Type: None (disables error dialog)
- Type: RTL (to use default dialog)
- Type: Console (writes to console)
- Type: MessageBox
- Type: MS Classic
- Type: EurekaLog
- Type: Bug report view (EurekaLog detailed)
- Type: "Enter steps to reproduce"
- Type: HTML page
- Type: System Log
- Type: WER (Windows Error Reporting)
- Option to enter "step to reproduce" text
- Option to specify e-mail
- Option to terminate or restart application
- Option to send or not send bug report
- Option to attach or not attach screenshot
- Custom "Help" button
- Custom "Support" link
- Customizable auto-close
- Can use icon and name of host application
- Auto-open and auto-position source code file in IDE by double-clicking on call stack items
- DPI-awared
- Localizable
- Unicode and RTL support
- Full customization

Sending
- Type: mailto: protocol
- Type: Simple MAPI
- Type: MAPI
- Type: SMTP Client
- Type: SMTP Server
- Type: HTTP upload (custom script)
- Type: FTP upload
- Type: FogBugz
- Type: Mantis
- Type: BugZilla
- Type: WER (Windows Error Reporting)
- Compression and encryption (all except mailto:)
- Full SSL/TLS support (SMTP, HTTP, web trackers, WER)
- Your code can supply custom web-fields for web trackers and HTTP upload
- Can send via multiple methods
- Visual feedback during sending
- Unicode support
- Backward feedback feature: report if bug was fixed, ask for more information (HTTP and web tracker software only)
- Full customization

Debug information formats
- Type: EurekaLog
- Type: .map

- Type: Turbo Debugger (TD32/TDS)
- Type: DLL exports table (heuristic)
- Type: DBG
- Type: PDB
- Type: JEDI (.jdbg/JCL)
- Type: madExcept (experimental)
- Auto-downloading system debug information
- Fully customizable
- Encrypted even with no password
- Can be protected by password (encrypted reports can be decrypted by Viewer)

Localization
- Full support for localization software
- No localization tool is required
- Resourcestrings
- Translate-function (GetText-style)
- Customize text right in project options
- Named collections for translated texts
- Unicode support

Tools
- Bug Reports Viewer
- Address Lookup
- Error Lookup
- Threads Snapshot
- Executable Modules Analyzer
- Standalone Settings Editor

Viewer
- EurekaLog Viewer Tool to view bug reports
- Viewer can decrypt encrypted reports
- Report printing
- Viewer can work as "viewer" or as bug tracker software (collect bug reports into database)
- Supports plain-text or FireBird database
- Auto-download reports from folder or e-mail account
- Can eliminate duplicate bug reports
- Shows screenshots inside bug reports
- Shows additional files inside bug reports
- Shell integration
- Support user accounts (for FireBird database only)

Misc.
- Supports generics in your code (debug information)
- FastMM compatibility
- Shared memory manager compatibility
- Support for any 3rd party memory manager (some EurekaLog features may be disabled)
- Multi-monitor support (screenshots)
- Option to reduce executable file size (remove relocs)
- Option to detect executable file changes
- Command-line compilation support
- FinalBuilder support
- Vista and UAC friendly

3.2 How to buy

Buy now

We accept eight types of payment: Secure Online (Credit Card), Fax, Phone, Wire Transfer, Mail / Check / Money Order, PayPal, Purchase Order and Bank Transfer. You can also order via our world wide resellers (see the full list), so EurekaLog may be available in your local currency and/or by other payment methods.

There are three license types:

- **Single** license - single developer.
- **Company** license - unlimited company developers at one geographical address.
- **Corporate** license - unlimited company developers at unlimited geographical addresses.

Please refer to complete license text 639 for more information about license differences.

EurekaLog also have three editions:

- **Trial** - it's fully functional edition with (*one* and *only*) additional limitation: any application, which is compiled with Trial edition of EurekaLog will expire after 30 days. There will be error message box after 30 days and application will exit. Trial itself may be used for infinite time. This edition can be used to evaluate EurekaLog. You can not use this edition for any commercial development.
- **Professional** - it's fully function edition. This is minimum edition to do actual work except evaluating (including commercial development).
- **Enterprise** - it's the same as Professional edition, except it additionally offers full source code of EurekaLog.

See Features 10 topic for more information about license's differences.

Delivery

When you buy EurekaLog, you IMMEDIATELY receive download instructions via email. An account on our site will be created for you. You can login in our area for the registered customers here by using your e-mail address as login (you can find your password in the registration e-mail). EurekaLog do not use any license information except your account (login/password pair). I.e. your account data is your license information.

Get access to earlier versions

EurekaLog 7 is the current version of EurekaLog. With EurekaLog, you also get free access to licenses for older versions – EurekaLog 4, EurekaLog 5, and EurekaLog 6. Download links to installers of earlier versions will be available in your control panel after purchase. Please note that old versions are no longer developed nor supported. They are provided only for backward compatibility purposes. You can use them, if you need support old systems (Delphi 3, Windows 95, Windows 98, Windows ME, Windows NT).

License

The EurekaLog license is royalty-free.
See the complete license text 639.

Guarantee

FULL MONEY BACK GUARANTEE.
All EurekaLog versions are covered by 60-day money-back guarantee.
If you are not satisfied with your purchase for any reason, just ask for your money back and you will be refunded.

Update Policy

- **All updates** (to minor and major versions) **within 1 year** are fully free.
- **All updates** (to minor and major versions) **after first year** are sold with 50% discount (for 1 more year).
- Updates to a **different license type of the same major version** are sold at only price difference (example: from Single Professional version to Company Enterprise).

Note: Please note that update policy stated above is applicable to current EurekaLog version only (i.e. EurekaLog 7). For owners of previous EurekaLog versions - please, refer to upgrade policy for your EurekaLog version.

Support

All EurekaLog versions are covered by full and unlimited support 16.

Partner discounts

Ordering EurekaLog you receive discounts for other partner products (see partners page for further details).

Buy now

3.3 Support

All EurekaLog versions are covered by full and unlimited support.

Where?

There are many ways to contact EurekaLog's team:

- support sending form: http://www.eurekalog.com/support.php
- forums: http://news.eurekalog.com/
- knowledge base (KB): http://support.eurekalog.com/index.php?/Knowledgebase/List/
- sending e-mail to support@eurekalog.com
- using our support system directly
- there are also IM contacts, listed on support form page

The best option is to use support sending form (http://www.eurekalog.com/support.php**)**.

That's because you can select type of your request, so it will go directly to appropriate category in our internal support system (the very similar way is to use our support system - it provides a little more detailed options, but setting them wrong can slow down your ticket processing).

You can use our forum, if you want to discuss your issue in public (all other ways of conversation keeps your messages private, not accessible by other users) or if you want to hear opinions from other customers of EurekaLog (and not only from EurekaLog team). In all other cases it is best to select another way.

The sending e-mail (while looking very convenient) is not a best choice. That's because e-mails must be sorted and inserted into queues manually. So, if you send us a e-mail with support request, it first needs to be redirected to appropriate ticket's queue and only then answered by staff. Other reason: you may forget to specify about what version of EurekaLog you're talking, so your request may be placed in wrong queue (like Delphi/C++ Builder instead of .NET), so it may takes days (spend in strange conversation), until we realize mistake. In other words, if you want to get reply faster - it is better to use support sending form and specify proper options.

Our internal support system also have external interface, so you can use it directly. It provides you most options, but probably it is too much for majority of our customers. It can be also used to troubleshot problems with your support tickets (see below).

Of course, you can also use IM client to ask questions. But it is better to use it to ask short questions, which requires simple answer. Like "Is there any plan for XXX?". Such answer can be answered very fast. It is not a good idea to ask questions like "I'm trying to do XXX. I do YYY and ZZZ. I'm getting AAA instead of BBB. What am I doing wrong?". It is best to send this question via support sending form (it is also a good idea to attach screenshots and demo).

What to write?

It is very important part, as it may greatly speed up/slow down processing of your request.

First, simple rules about choosing options for your ticket (if you use support sending form): select appropriate category for your request.

- If you have question about prices, warranty, licenses, upgrades, your registered previous account/e-mail, etc, etc - you should place it to "Sales" category.
- If you have question about EurekaLog usage, how to do XXX, does it support YYY, problems, bugs, etc, etc - you should place it to "Support" category.
- Place your question in "General" category only if it is really a general question. Like "Are there any plans for EurekaLog for PocketPC"? In other case, your ticket will be redirected to other category, and you just lose your time waiting.

Other important part is the message itself.

PLEASE, specify your versions and editions of EurekaLog, IDE and Windows. We do not ask you for license details, but remember that support staff do not have telepathic abilities and they can't read your thoughts. If you don't point this, then you will slow down processing, as the very first reply may be like this: "Excuse me, are we talking about Delphi or C++ Builder here?". BTW, it is good idea to reinstall EurekaLog, using the latest available RC version. That's because your issue may be already fixed. You may also send a request and check a new version while request is processed.

Of course, your message should be quite clear about what you want us to do.

Other piece is demo or steps to reproduce. Many of our clients think: "gosh, it is so simple, just do XXX and you'll see what I mean". Except when we do this on our machines - nothing happens. That's because client uses some option or event handler. So it is a good idea to reproduce your issue in just empty VCL application with one TButton and "raise Exception.Create('Test');" in its OnClick. If you try to do that, you may often unable to get test application behave like desired. And if you look deeper - you may notice differences between test application and your application (options, events, etc), that caused your issue. That way you'll be able to solve it by yourself (BTW, we recommend to use import/ export buttons to transfer your EurekaLog's settings between projects). Doing a little work before sending question can save you days of conversation.

Of course, there can be cases, when you're unable to reproduce your issue in new application. In that case just mention it: "I wasn't able to reproduce it in a test project" and don't forget to attach your project's settings, screenshots, etc - anything, that can give a hint.

Hint: instead of screenshooting every page of your EurekaLog's options, you may just export your settings to .eof file and send it to us (there is "Export" button in EurekaLog's options dialog).

Well, it is totally fine to state your requests as "I've assigned exception event, but it is not getting called. What am I doing wrong?", but be prepared to long conversation in that case.

Using native languages?

If you don't speak English, then you can write in your native language. EurekaLog staff will use some translator to translate your request and answers to appropriate language. But it is not recommended. If you only know English a little, you may say your request as you can in English and repeat it on your native language. That's because it's hard to understand machine translations. So if your request was short-formulated and was badly translated by auto-translator - we'll simply not understand you. So attaching your request in bad, but still "human-produced" English may greatly help.

Some of you may write in native language even if you're capable of speaking English - if you know, that someone from EurekaLog team speaks on that language too. Well, this is still a not so good idea. As your request may be answered by different people, and sometimes we discuss certain issues together, so it is good to keep conversation in English, so everyone can read it (without need to use machine translator).

So there can be cases of some misunderstanding. If you think that somebody don't get your actual request - please, try to explain in other words or in more details.

I'm not getting any response?

Unfortunately, sometimes this happens. Really, we may miss your reply and so your thread may be forgotten, but we can assure you, that this is really a rare case. In most other cases there is lost e-mail or similar case. If you have not received expected reply in reasonable amount of time (say, a week) - send your message again!

If you use e-mails, then there can be chances that reply was marked as spam or was rejected due to suspicious attachment. Please, check you trash/spam folder (or similar) in your e-mail client. Do not hesitate to send reply again, if you suspect that your reply can be lost.

Please note, that we use GMail for delivering e-mails, so be sure that your messages do not contain illegal attachments (you can rename files or pack them into 7z-archive).

If you use our support system (note, that your e-mails are redirected to our support system too) - then it is the best case, as you can check status of your request in any time. First, note, that every time your request is registered, you should get automatic response like this:

> Thank you for contacting us. This is an automated response sent to you in order to confirm the receipt of your message. We will attend to your ticket as soon as possible. We've listed the details of the ticket you created below for your records. When replying, please keep the ticket's ID in the subject to ensure that your replies are tracked correctly.
>
> **Ticket ID**: *your-ticket-ID-here*
> **Subject**: *your-subject-here*
> **Department**: Delphi-Support
> **Priority**: Low
> **Status**: Open
>
> You can check the status of or reply to this ticket online at: http://support.eurekalog.com/
> Please use the following credentials when accessing your account with us:
>
> **Email**: *your-e-mail-here*
> **Password**: *your-password-here*

If you don't receive this reply (and it is not in your "spam" folder) in few hours - resend your request immediately! You may also try to use different contact method.

So, if you didn't receive any reply for long period of time - you can check the entire history at http://support.eurekalog.com/. Just enter your e-mail and password, which were provided in automatic reply. You'll see all your and staff replies, so you can check if all replies were delivered correctly.

If you think there is a problem with delivering on our side - do not hesitate to make complaints.

Part IV

4 Quick start tutorials

EurekaLog is really easy to use!

Just install it 20 and enable it 33 for your project! **You don't need to write a single line of code for most application types**.

Take a look at basic procedures 45 topic, if you're looking for something more interesting.

See also for concept and theory:

- EurekaLog's basics 38
- Terms 40

4.1 Installation

This article is part of Quick start tutorials 20 series. Please refer to different guides for the following specific cases:

- You want to install EurekaLog for AppWave/All-Access versions of Delphi/C++ Builder 31
- You want to install EurekaLog for limited IDE edition (Personal, Turbo, Starter) 23
- You want to install EurekaLog on machine without IDE installed 31
- You want to install EurekaLog for different user account 31
- You want to install EurekaLog for virtualized IDE 31

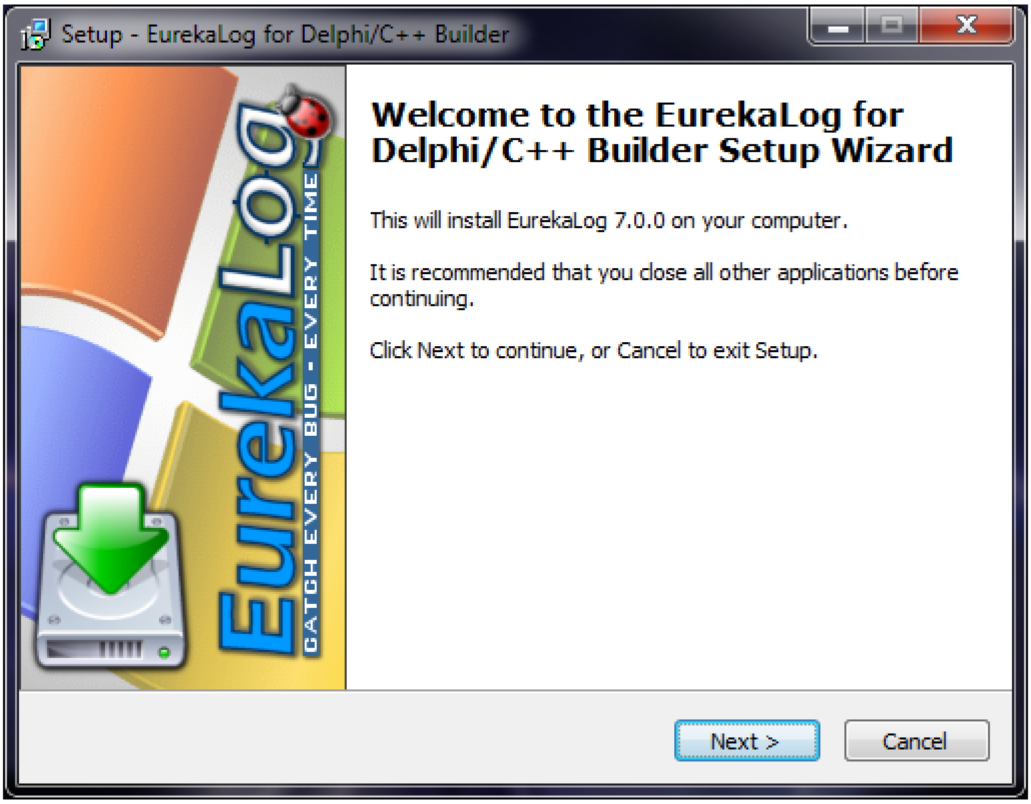
Otherwise (i.e. you're typical user, you want to install EurekaLog for classic Delphi/C++ Builder IDE) - continue reading.

Download the latest version of EurekaLog from our site.

- You can use trial version to evaluate EurekaLog.
- If you've already bought EurekaLog - you can download your version from our area for the registered customers.

See also edition differences 10.

1. Before installing EurekaLog, **you must close all Delphi/C++Builder/RAD Studio/ AppMethod instances**.
2. Run the EurekaLog installation program:



Setup - EurekaLog for Delphi/C++ Builder

Welcome to the EurekaLog for Delphi/C++ Builder Setup Wizard

This will install EurekaLog 7.0.0 on your computer.

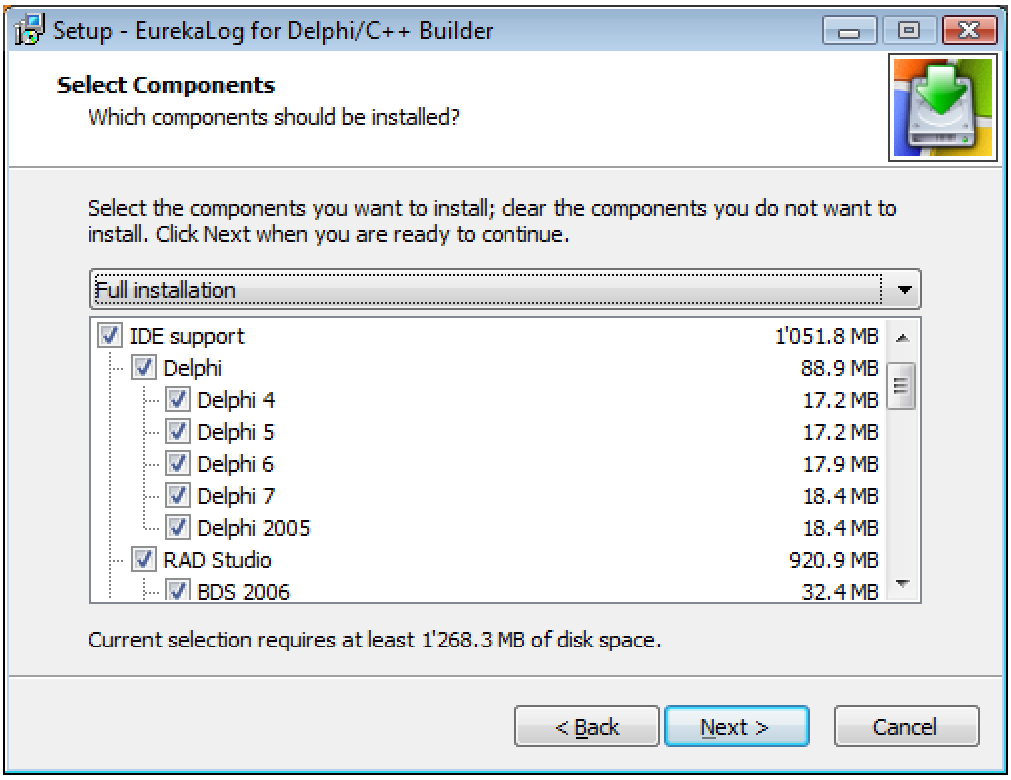
It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Installation wizard started

Note: installer will ask you to uninstall old EurekaLog versions before continue. You can continue without uninstallation, if you want to have both EurekaLog versions installed.

3. Select one or more Delphi/C++Builder versions you wish to install:



Setup - EurekaLog for Delphi/C++ Builder

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ IDE support	1'051.8 MB
☑ Delphi	88.9 MB
☑ Delphi 4	17.2 MB
☑ Delphi 5	17.2 MB
☑ Delphi 6	17.9 MB
☑ Delphi 7	18.4 MB
☑ Delphi 2005	18.4 MB
☑ RAD Studio	920.9 MB
☑ BDS 2006	32.4 MB

Current selection requires at least 1'268.3 MB of disk space.

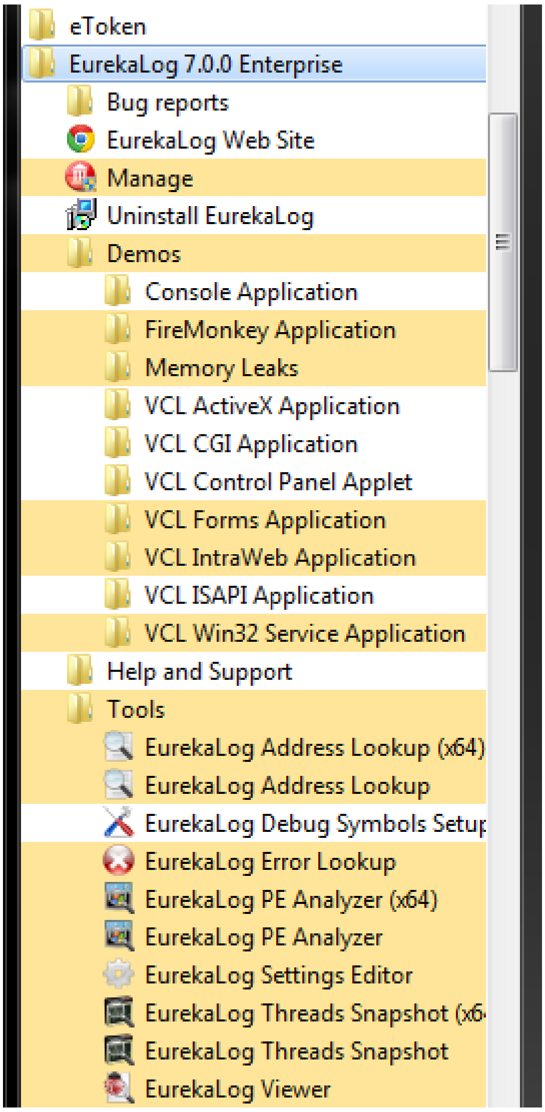
Features selection

Installer will show you only available IDEs - based on your installed IDEs as written in system registry. You can check/uncheck appropriate checkboxes to install or don't install IDE support (set of .dcu files, .bpl, etc). There is individual set of .dcu/.obj/.hpp/.bpl files for each IDE.

Don't see your IDE version? 596

Source files (if available in your edition of EurekaLog) are installed by using separate checkbox "Source code" below. Source files are identical for all IDEs. There is only 1 set of files.

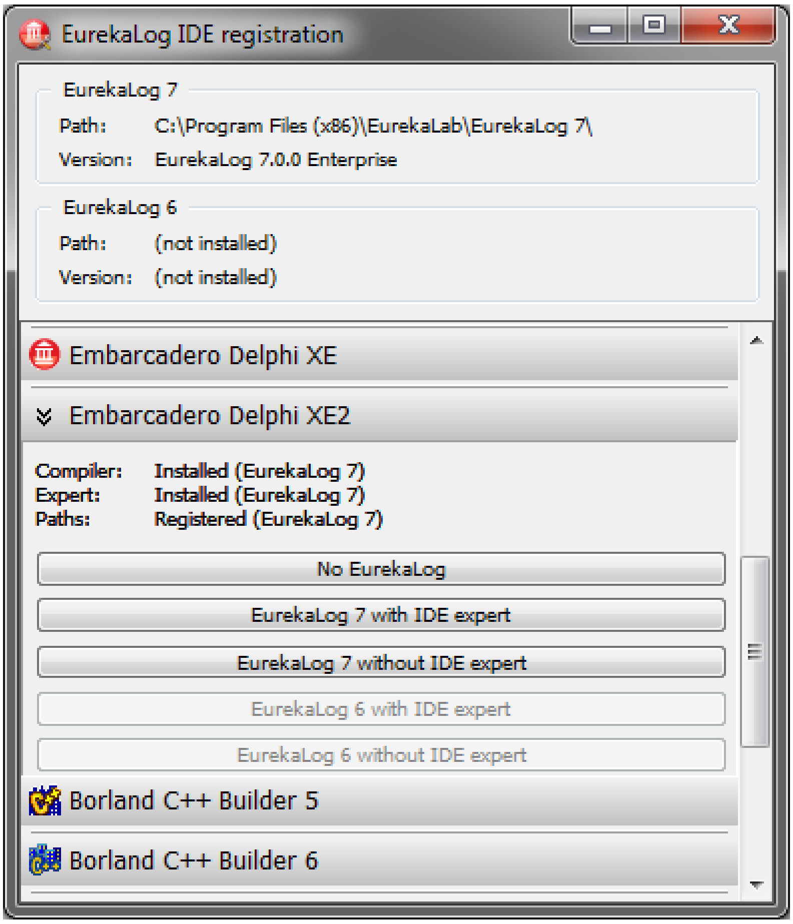
4. Follow installation wizard's steps until the installation will be complete.
5. EurekaLog is ready to use. You can find links in Start Menu:



Installed EurekaLog in Start menu

6. Run Delphi or C++ Builder and enable EurekaLog for some or all of your projects 33.

Note: you can use "Manage" start menu item to enable/disable EurekaLog in particular IDEs after installing.


EurekaLog IDE registration

EurekaLog 7
Path: C:\Program Files (x86)\EurekaLab\EurekaLog 7\
Version: EurekaLog 7.0.0 Enterprise

EurekaLog 6
Path: (not installed)
Version: (not installed)

Embarcadero Delphi XE
Embarcadero Delphi XE2

Compiler: Installed (EurekaLog 7)
Expert: Installed (EurekaLog 7)
Paths: Registered (EurekaLog 7)

No EurekaLog
EurekaLog 7 with IDE expert
EurekaLog 7 without IDE expert
EurekaLog 6 with IDE expert
EurekaLog 6 without IDE expert
Borland C++ Builder 5
Borland C++ Builder 6

"Manage" tool from Start menu

Each installed and supported IDE will be present by roll-up category panel. You can expand category for each IDE by clicking on it. You'll see info about EurekaLog in this IDE and available options.

See also:
- Problems? 596
- What's next? 33
- Where to find EurekaLog? 603
- Using EurekaLog with Delphi Personal/Turbo/Starter editions 23
- Installing EurekaLog for non-administrative user accounts 31
- Installation for application virtualization (AppWave) or without IDE installed 31

4.1.1 Using EurekaLog with Delphi Standard/Personal/Turbo/Starter editions

Borland/CodeGear/Embarcadero has several very limited editions of Delphi and C++ Builder which are designed for beginners and IT hobbyists. This article discusses possible pitfalls in using EurekaLog on these IDEs:
- Delphi 7 Personal
- Turbo Delphi/C++Builder Explorer
- Turbo Delphi/C++Builder Professional
- Delphi/C++Builder XE/XE2/XE3/XE4/XE5/XE6/XE7/XE8 Starter

Common information

EurekaLog installer contains precompiled .dcu and .obj files. These files are used when you compile your applications. Therefore, there is no requirement to has command-line compiler in IDE.

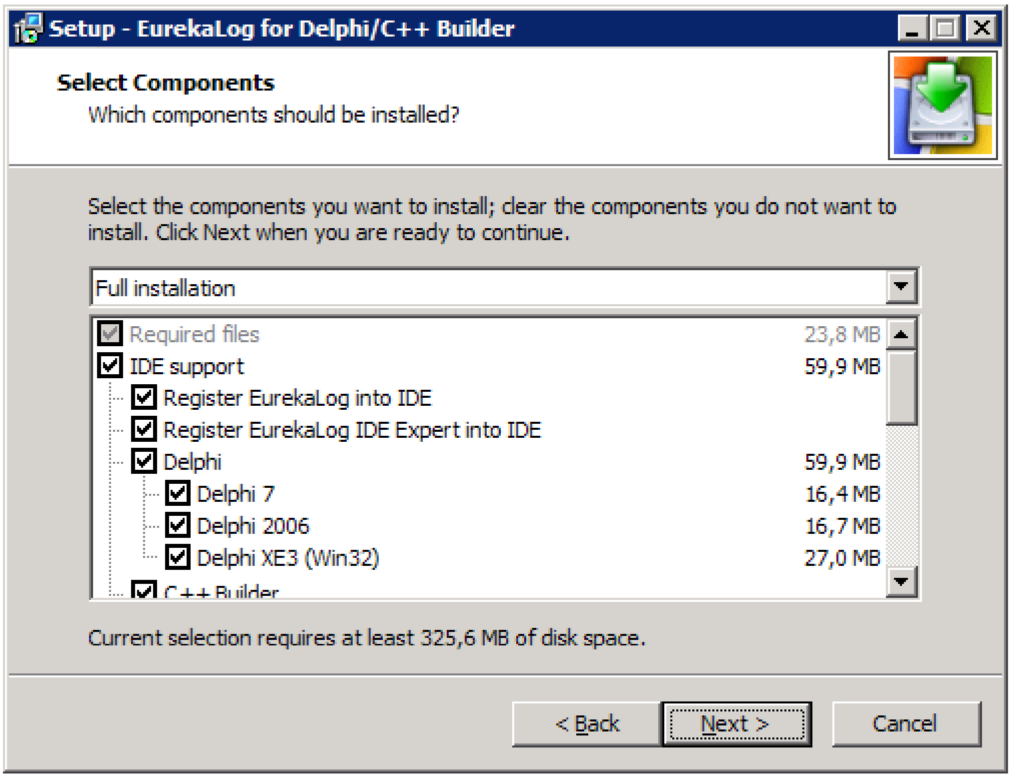
Note: EurekaLog installer also contains .pas files, but they are used only as reference. .pas files are not used for projects compilation by default. Owners of Enterprise (full source-code) edition of EurekaLog can delete .dcu/.obj files and use .pas files instead. Users of Trial and owners of Professional edition are not able to compile .pas files (because they contain only interface sections, but not implementation).

EurekaLog is compiled against the latest service/update pack for each IDE, so be sure to install the latest available update for your IDE before installing EurekaLog. You can get the latest update for your IDE from your EDN account under "My registered user downloads" section.

EurekaLog requires processing of compiled executables (post-processing). Typically this is done by installing IDE extension to handle this issue automatically. See for more info 38. Therefore your IDE must have ability to install 3rd party extensions.

However, there is a second way to achieve this - by calling EurekaLog command-line tools manually. You don't need to install IDE extensions for this way. More on this below.

There are no limitations on number of supported IDEs in EurekaLog, so you can install EurekaLog for, say, Delphi 7 Personal, Turbo Delphi and Delphi XE3 Starter simultaneously.



Setup - EurekaLog for Delphi/C++ Builder

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ Required files 23,8 MB
☑ IDE support 59,9 MB
- ☑ Register EurekaLog into IDE
- ☑ Register EurekaLog IDE Expert into IDE
- ☑ Delphi 59,9 MB
  - ☑ Delphi 7 16,4 MB
  - ☑ Delphi 2006 16,7 MB
  - ☑ Delphi XE3 (Win32) 27,0 MB
- ☑ C++ Builder

Current selection requires at least 325,6 MB of disk space.

Delphi 7 Personal, Turbo Delphi Explorer and Delphi XE3 Starter in EurekaLog installer

This will install all necessary files for the selected IDEs:

- Set of precompiled .dcu files for each selected IDE
- IDE expert and run-time package for each IDE (.bpl)
- Command-line tools for each IDE (ecc32.exe/emake.exe)
- Single set of common source files (.pas)

All these files are the same as files for Delphi Professional/Enterprise/Architect/Ultimate. There are no differences in installing EurekaLog for, say, Turbo Delphi and Delphi 2006 Professional.

Notes:

- Personal, Turbo and Starter IDE's editions can be installed on the single machine.
- You can not install two Turbo's or two Starter's on the same machine. So, you can not have Delphi and C++ Builder personalities for the same IDE.

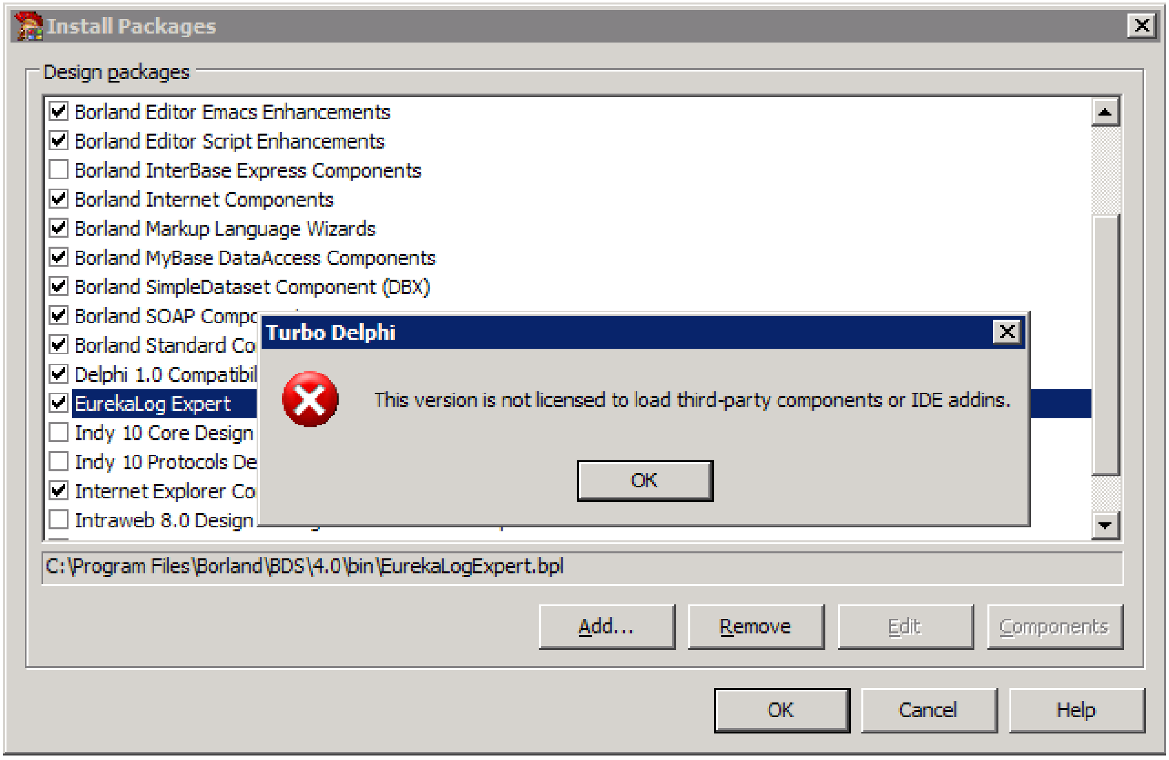
- It's important to not confuse Turbo Explorer and Turbo Professional editions of Delphi and C++ Builder. These IDEs has significantly different limitations for installing EurekaLog.

Turbo Professional and Starter editions

Both Turbo Professional and Starter editions of Delphi and C++ Builder allow to install and use 3rd party IDE extensions. Therefore you don't need any special actions to use EurekaLog in these IDEs. Just install EurekaLog as usual. You will see EurekaLog menu items in IDE. All your project will be post-processed automatically.

Turbo Explorer and Personal editions

Turbo Explorer edition do not allow you to install 3rd party IDE extensions. EurekaLog will install all necessary files (as outlined above), however IDE will refuse to load EurekaLog's IDE expert:



☑ Borland Editor Emacs Enhancements
☑ Borland Editor Script Enhancements
□ Borland InterBase Express Components
☑ Borland Internet Components
☑ Borland Markup Language Wizards
☑ Borland MyBase DataAccess Components
☑ Borland SimpleDataset Component (DBX)
☑ Borland SOAP Compo
☑ Borland Standard Co
☑ Delphi 1.0 Compatibil
☑ EurekaLog Expert
□ Indy 10 Core Design
□ Indy 10 Protocols De
☑ Internet Explorer Co
□ Intraweb 8.0 Design

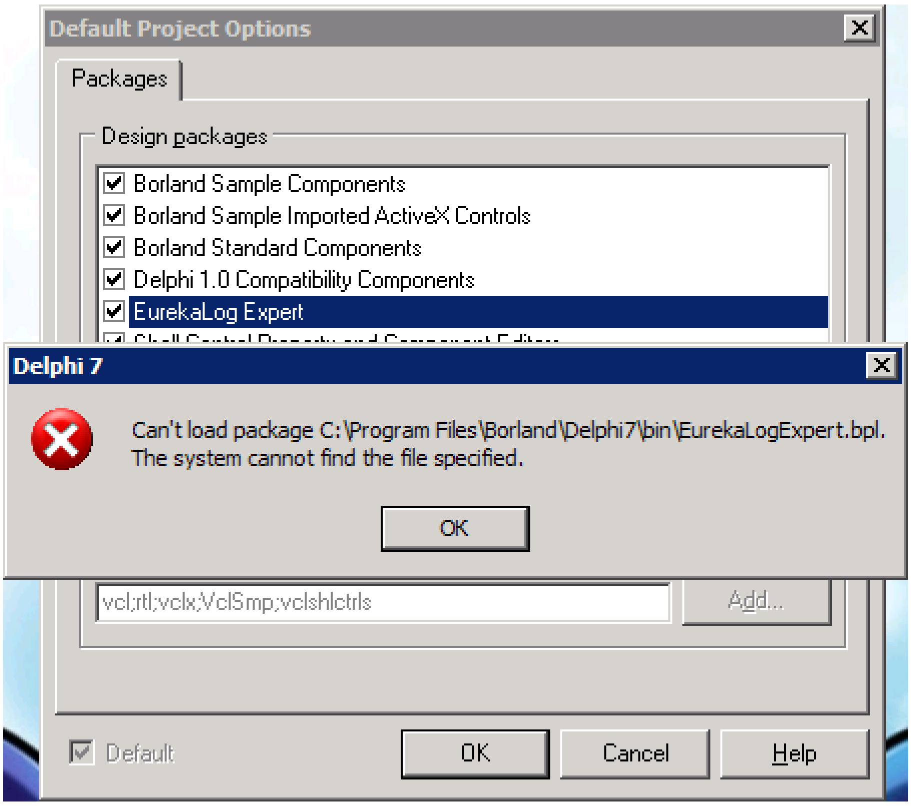
This version is not licensed to load third-party components or IDE addins.

C:\Program Files\Borland\BDS\4.0\bin\EurekaLogExpert.bpl

Turbo Explorer refuses to load EurekaLog IDE expert due to licensing limitations

Therefore EurekaLog menu items will be unavailable. Automatic post-processing of your projects (required for EurekaLog to function) will be disabled.

Delphi 7 Personal edition allows you to install 3rd party IDE extensions. However, it lacks required components (.bpl packages) for EurekaLog IDE Expert.



☑ Borland Sample Components
☑ Borland Sample Imported ActiveX Controls
☑ Borland Standard Components
☑ Delphi 1.0 Compatibility Components
☑ EurekaLog Expert

Can't load package C:\Program Files\Borland\Delphi7\bin\EurekaLogExpert.bpl.
The system cannot find the file specified.

vcl;rtl;vclx;VclSmp;vclshlctrls

☑ Default

Delphi Personal reports that EurekaLog IDE expert is unable to load due to missing components

(This error message may be confusing, as it reports "missing file" issue as if it is related to EurekaLogExpert.bpl file. This is wrong interpretation: EurekaLogExpert.bpl file exists. This error message says that EurekaLogExpert.bpl is unable to load because its required files are not found - such as soaprtl70.bpl, etc.)

Again, IDE features of EurekaLog will be inaccessible (unless you have missed Delphi 7 .bpl files from somewhere else).

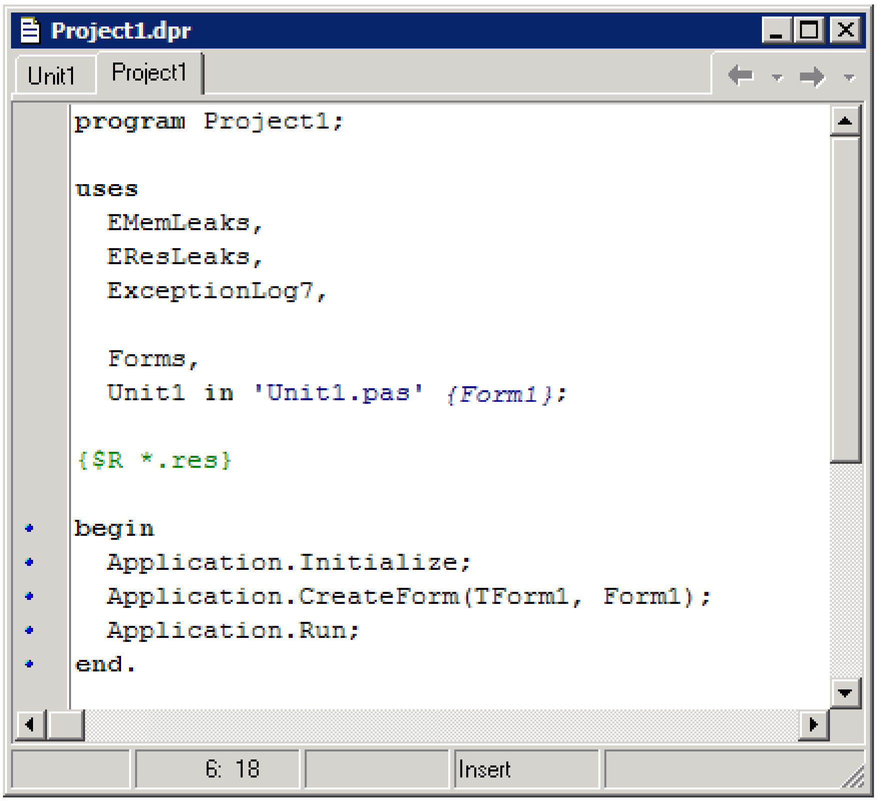
Note: You can remove EurekaLog IDE expert by using "Start/Programs/EurekaLog 7/ Manage" menu item and clicking on "Install EurekaLog 7 (without IDE expert)" button under your IDE name.

Workaround for Turbo Explorer/Personal editions

The workaround is explained in great details in our help here 421 (specifically, see "Post-processing without (re)compilation 426" section). Let's do only a quick overview in this article and provide a practical example.

The example will use Delphi 7 Personal, but the same things should be applicable to other IDEs.

First, you need to create a new application and add necessary EurekaLog's units to your application. Use "Project/View Source" command to open .dpr file and add at least `EMemLeaks`, `EResLeaks` and `ExceptionLog7` units to your uses clause:

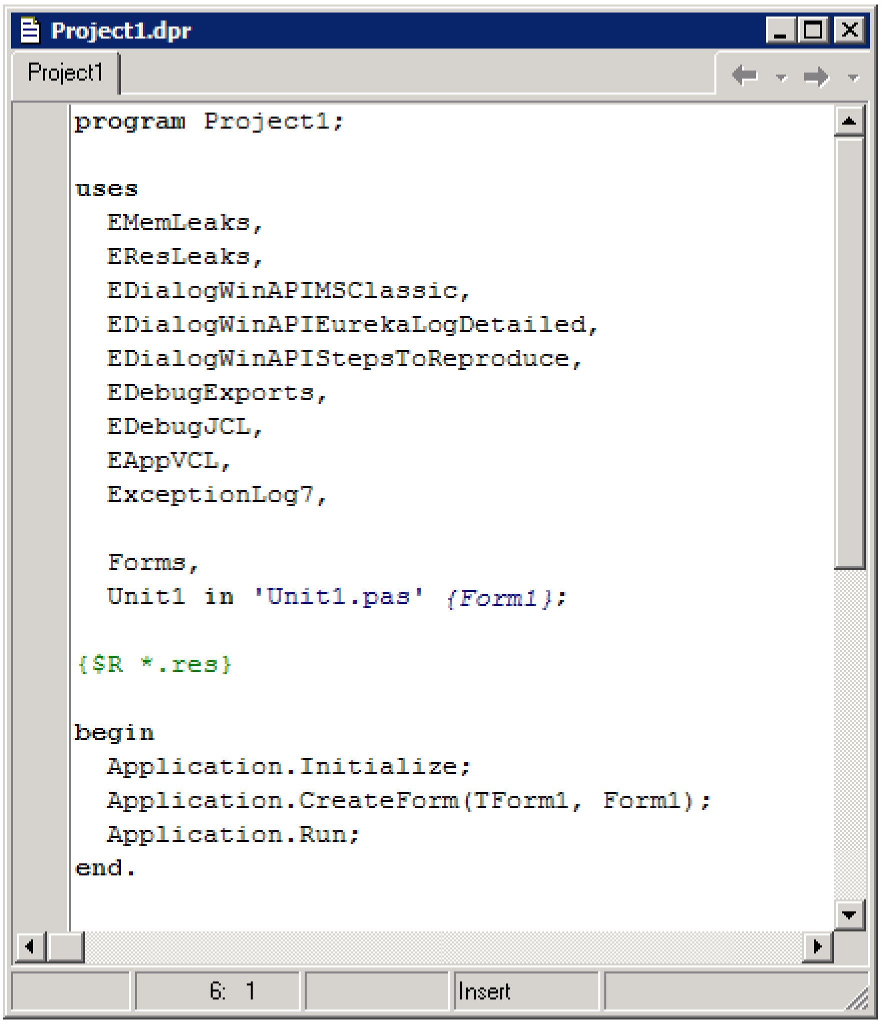


```
program Project1;

uses
  EMemLeaks,
  EResLeaks,
  ExceptionLog7,

  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Adding EurekaLog code to your project

Important note: order of units is important. These units must be listed first - as shown above.

These are the minimum units for EurekaLog to work correctly. However, you may want to add more units - such as `EDialogWinAPIMSClassic` for error dialog in MS Classic style, `EAppVCL` for hooks on Forms unit, `EDebugExports` to show functions from DLLs in call stacks, `ESendAPIMantis` to submit bug reports to Mantis bug tracker, etc.



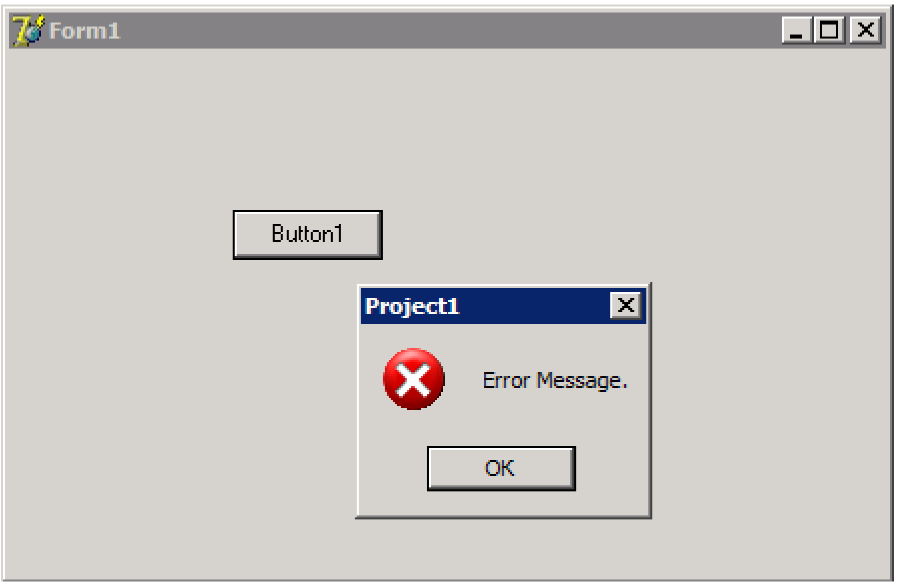
```
program Project1;

uses
  EMemLeaks,
  EResLeaks,
  EDialogWinAPIMSClassic,
  EDialogWinAPIEurekaLogDetailed,
  EDialogWinAPIStepsToReproduce,
  EDebugExports,
  EDebugJCL,
  EAppVCL,
  ExceptionLog7,

  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Typical units for VCL forms application

Second, you need to setup project options for debugging. Please see "Configuring project 58" article. That article will explain what options are required - you should enable them for your project.

Third, you need to create configuration for EurekaLog. Use "Start/Programs/EurekaLog 7/ Tools/Settings Editor" menu item to launch standalone editor for EurekaLog settings. Setup EurekaLog options as you desire and click on "Save" button to save all options to .eof file.

Now you can place a button in your application to raise exception (to test EurekaLog) and compile your application. It's better to make a "build" for first time, not just "compile". Your project will be compiled with EurekaLog and debugging options, but EurekaLog will be in disabled state. You can run your application, it will be fully functional, but exceptions will be handled by RTL/VCL, not by EurekaLog.



Error Message.

Application with disabled EurekaLog

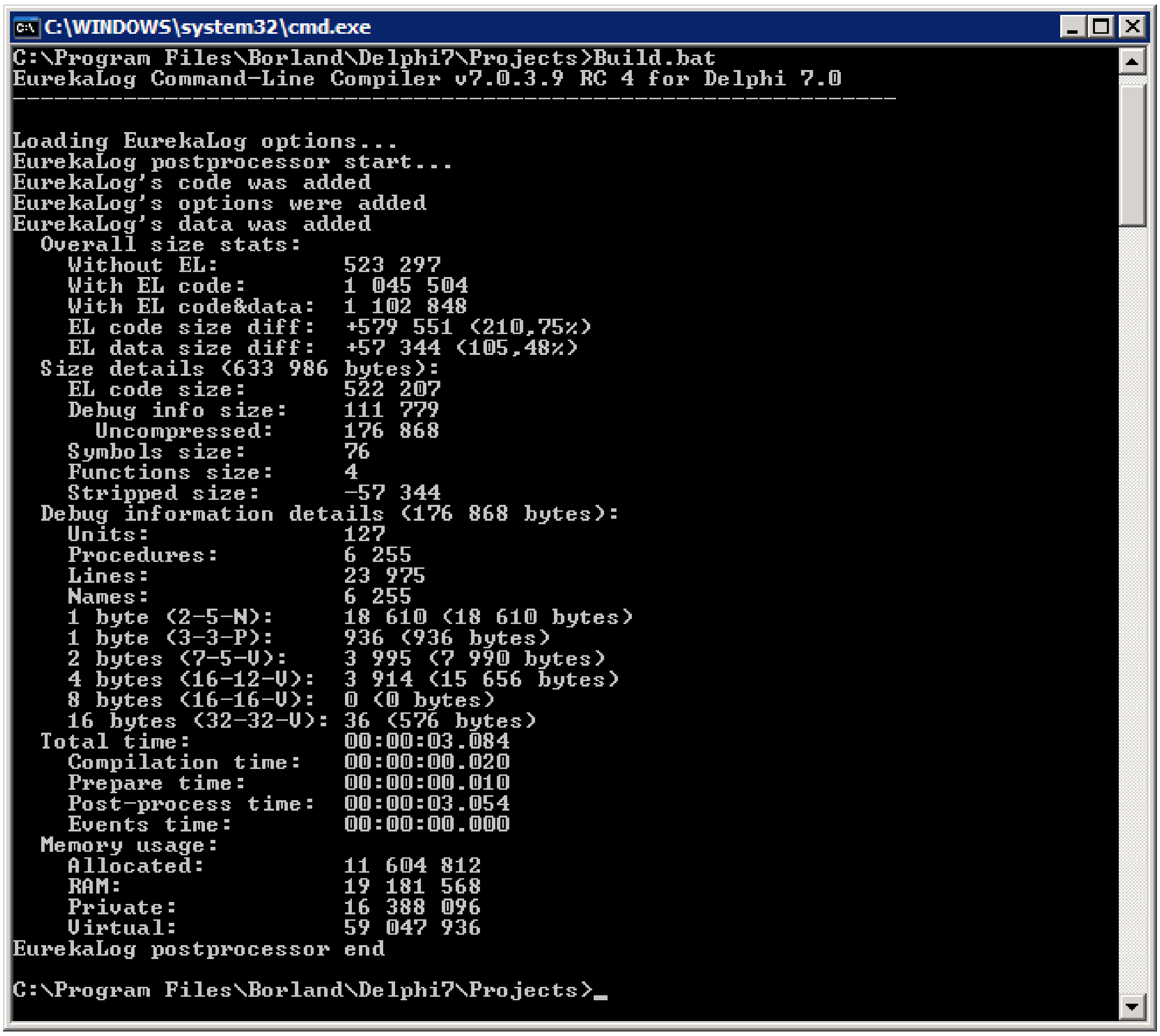
Final step is to post-process your executable 426. This step should inject debug information and EurekaLog's options into your executable. To do that - create text file with .bat extension (say, "Build.bat") and with the following content:

```
@echo off
"C:\Program  Files\Borland\Delphi7\Bin\ecc32.exe"  "--el_alter_exe=C:\Program  Files
\Borland\Delphi7\Projects\Project1.dpr;C:\Program    Files\Borland\Delphi7\Projects
\Project1.exe" "--el_config=C:\Program Files\Borland\Delphi7\Projects\Project1.eof"
pause
```

Notes:

- Adjust file names and paths as necessary;
- You can also use relative file paths;
- Exact location of .bat file does not matter - as long as all file paths point to correct files;
- Typically you place .dpr/.eof/.bat files in the same folder;
- Last command (pause) is optional.

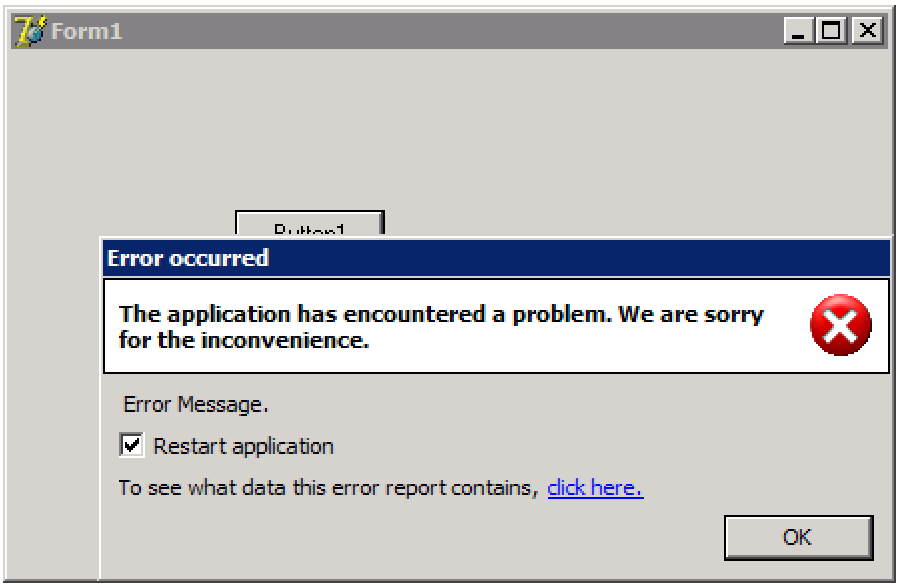
Now you have to run this .bat file each time after compiling your project.



```
C:\Program Files\Borland\Delphi7\Projects>Build.bat
EurekaLog Command-Line Compiler v7.0.3.9 RC 4 for Delphi 7.0
----------------------------------------------------------------

Loading EurekaLog options...
EurekaLog postprocessor start...
EurekaLog's code was added
EurekaLog's options were added
EurekaLog's data was added
  Overall size stats:
    Without EL:           523 297
    With EL code:         1 045 504
    With EL code&data:    1 102 848
    EL code size diff:    +579 551 (210,75%)
    EL data size diff:    +57 344 (105,48%)
  Size details (633 986 bytes):
    EL code size:         522 207
    Debug info size:      111 779
      Uncompressed:       176 868
    Symbols size:         76
    Functions size:       4
    Stripped size:        -57 344
  Debug information details (176 868 bytes):
    Units:                127
    Procedures:           6 255
    Lines:                23 975
    Names:                6 255
    1 byte (2-5-N):       18 610 (18 610 bytes)
    1 byte (3-3-P):       936 (936 bytes)
    2 bytes (7-5-V):      3 995 (7 990 bytes)
    4 bytes (16-12-V):    3 914 (15 656 bytes)
    8 bytes (16-16-V):    0 (0 bytes)
    16 bytes (32-32-V):   36 (576 bytes)
  Total time:             00:00:03.084
    Compilation time:     00:00:00.020
    Prepare time:         00:00:00.010
    Post-process time:    00:00:03.054
    Events time:          00:00:00.000
  Memory usage:
    Allocated:            11 604 812
    RAM:                  19 181 568
    Private:              16 388 096
    Virtual:              59 047 936
EurekaLog postprocessor end

C:\Program Files\Borland\Delphi7\Projects>_
```

Successful post-processing by ecc32.exe tool

Run your application now. If you do everything correctly - EurekaLog will be active, and exceptions will be handled by EurekaLog.



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Error Message.

☑ Restart application

To see what data this error report contains, click here.

Same application after post-processing: EurekaLog is active

Additional limitations

Unaccessible functions

Some auxiliary EurekaLog functions may be unaccessible due to missing units from Personal/Turbo/Starter edition.

Debugging

Obviously, you will not be able to debug your applications directly if you do manual post-processing (Delphi Personal and Turbo Explorer). That's because you have to compile your project, then call .bat file to post-process, then run the project. But this last step (running) will rebuild your project if you run it under debugger. So, you have the following options:

- (For EurekaLog) Compile -> post-process -> run outside of IDE
- (For debugging) Compile -> run under debugger
- (For both) Try to use `Run/Load process` command or `Run/Attach to process` command
- (For both) Upgrade your IDE

Note: this limitation is true only for older IDEs. Newer IDEs has build events for project, so you may call ecc32/emake from these build events 429.

See also:

- Installing EurekaLog 20

4.1.2 Installation for non-admin user account

Please, read normal installation instructions 20 first.

To install EurekaLog under limited user account (non-administrator) - please follow these steps:

1. Install EurekaLog as usual 20 - by launching installer under administrator account.
2. Use Start Menu / Programs / EurekaLog / Manage menu item to enable EurekaLog in your IDEs. Run it under the same administrator account (with elevation). Expand category for your IDE and click on "Install EurekaLog 7 (recommended)" button. This will copy necessary files. You should take this step only once.
3. Switch to limited user account.
4. Use Start Menu / Programs / EurekaLog / Manage menu item to enable EurekaLog in your IDEs. Run it under limited user account. Expand category for your IDE and click on "Install EurekaLog 7 (recommended)" button. This will perform per-user registration. Repeat this step for any other user account. You can register/unregister EurekaLog on per-user basis.

See also:

- Installation for application virtualization (AppWave) or without IDE installed 31
- Using EurekaLog with Delphi Personal/Turbo/Starter editions 23

4.1.3 Installation for AppWave or without IDE installed

General concepts

Application virtualization is software technology that encapsulates application software from the underlying operating system on which it is executed. A fully virtualized application is not installed in the traditional sense, although it is still executed as if it were. The application behaves at runtime like it is directly interfacing with the original operating system and all the resources managed by it, but can be isolated or sandboxed to varying degrees. In this context, the term "virtualization" refers to the artifact being encapsulated (application), which is quite different to its meaning in hardware virtualization, where it refers to the artifact being abstracted (physical hardware).

Examples of application virtualization technologies are: AppWave, Windows XP Mode, Wine, BoxedApp, Citrix XenApp, Novell ZENworks Application Virtualization, Endeavors Technologies Application Jukebox, Microsoft Application Virtualization, VMware ThinApp, etc.

Embarcadero AppWave

AppWave Store is an app store for Windows which uses application virtualization to avoid the hassles and risks of the usual Windows install process. The idea is that purchasing apps for Windows will be as simple as installing an app on a mobile using the Apple app store or Android Market. The underlying technology was developed to simplify deployment of Embarcadero’s tools. The All-Access subscription includes a tool box application that lets you run tools using “InstantOn”, which means no installation, just click and run.

In addition to the standard installation executables for AppWave applications, Embarcadero provides **AppWave Apps** versions of each product. Through the Embarcadero AppWave Store you also have access to many free apps from other parties. Apps simplify deployment and enable side-by-side versioning. When AppWave Browser is not running in Turbo mode, such as in a locked-down desktop environment, Apps are ideal because they do not affect system files or system registry settings.

AppWave Apps run within their own space without sharing DLLs or system-wide settings that could conflict with other applications. Application data, including configuration and license information, is stored locally on the hard disk, separate from the executable itself. AppWave Apps run on the local machine and are launched from the network using AppWave Browser. Apps launched from AppWave Browser benefit from application streaming because the application will begin to open before the product has been fully downloaded. However, if you need frequent access to an application, you should download and run the file locally, which is typically faster than running it from AppWave.

Technical issues

EurekaLog installer may be unable to gain access to settings and configuration of virtualized Delphi or C++ Builder IDE (such as IDEs with All-Access licenses, AppWave versions of classic IDEs). This depends on exact solution which you're using to virtualize applications. For example, AppWave version of Delphi is isolated from host machine/OS, so EurekaLog installer will not be able to "see" it during the installation.

Normally, you should try to run EurekaLog installer within virtualized environment, so EurekaLog installer may get access to IDE's settings. Otherwise EurekaLog installer would act as if there is no IDE installed (since it can not detect virtualized IDE).

Installation guide

There are two possible methods available:

Option A: Please, refer to documentation on your virtualization software to know about running two applications (i.e. Delphi/C++ Builder IDE and EurekaLog installer) within the same virtualized environment (sandbox). EurekaLog installer will be able to detect installed IDE and install and register EurekaLog properly when it is running in the same environment as IDE itself. Installation process should not be different from normal installation 20 (with respect to specifics of used application virtualization solution).

Option B:
(**You can also use this option to install EurekaLog on machine without any IDE installed, or when IDE installation is corrupted**)
You can instruct EurekaLog installer to install necessary files even there is no IDE installed. Of course, EurekaLog installer will not be able to properly register EurekaLog into (non-existent/invisible) IDE - you still have to perform manual installation 598. To force-install files for a specific IDE - please, run EurekaLog installer with the following command-line switches:

```
/Force_MV
```

where:
- *M* is either 'D' or 'C' - to indicate Delphi or C++ Builder IDE
- *V* is integer version of IDE. You can determinate integer version of your IDE by using this table 604.

For example, to install EurekaLog files for AppWave version of Delphi XE4:

```
EurekaLog7-Enterprise.exe /Force_D18
```

To install files for RAD Studio XE4 (i.e. both Delphi XE4 and C++ Builder XE4):

```
EurekaLog7-Enterprise.exe /Force_D18 /Force_C18
```

To install files for Delphi 7 with corrupted installation/registration:

```
EurekaLog7-Enterprise.exe /Force_D7
```

EurekaLog installer will act as if IDE was detected - i.e. installer will show you normal checkboxes for IDE's files during such installation. You should see a typical set of all necessary files (i.e. .dcu/.obj, .bpl, etc.) after installation - assuming that you did not cleared checkboxes.

Please note that EurekaLog's Manage tool will not work - because it will not see installed IDE(s). You have to register EurekaLog manually 598. That is (simplified instructions):
1. (Optional) Copy necessary files to location where your IDE (or your build tool) can see it.
2. Specify paths to EurekaLog files:
   a. Add %EUREKALOG%\Lib\%PLATFORM%\Release\%IDENAME%\, %EUREKALOG%\Lib\Common and %EUREKALOG%\Source\Extras\ folders to "Library Paths"/"Search Paths" option.
   b. Add %EUREKALOG%\Source\ folder to "Browsing Paths" option.
   c. Add %EUREKALOG%\Lib\Win32\Debug\%IDENAME%\ folder to "Debug DCU paths" option.
   d. Add %EUREKALOG%\Lib\Common\ folder to "Include paths" option.
3. (Optional) Register EurekaLog IDE expert by installing (registering) EurekaLogExpert.bpl package.
4. (Optional) Register EurekaLog Events component by installing (registering) EurekaLogComponent.bpl package.

Please, see manual installation 598 guide for more detailed information.

See also:
- Installation for non-admin user account 31
- Using EurekaLog with Delphi Personal/Turbo/Starter editions 23
- Manual installation 598
- Installation issues 596

4.2 How to use EurekaLog

This article is part of Quick start tutorials 20 series.

Important Note: dropping EurekaLog component on your form is not enough! You have to configure EurekaLog for your project. EurekaLog component is used to react to certain events in EurekaLog-enabled projects. The component will do nothing on its own.

Short answer:

To enable EurekaLog in your application:
1. Save your project.
2. Check "Activate EurekaLog" option.
3. Select application type.
4. Rebuild your project.

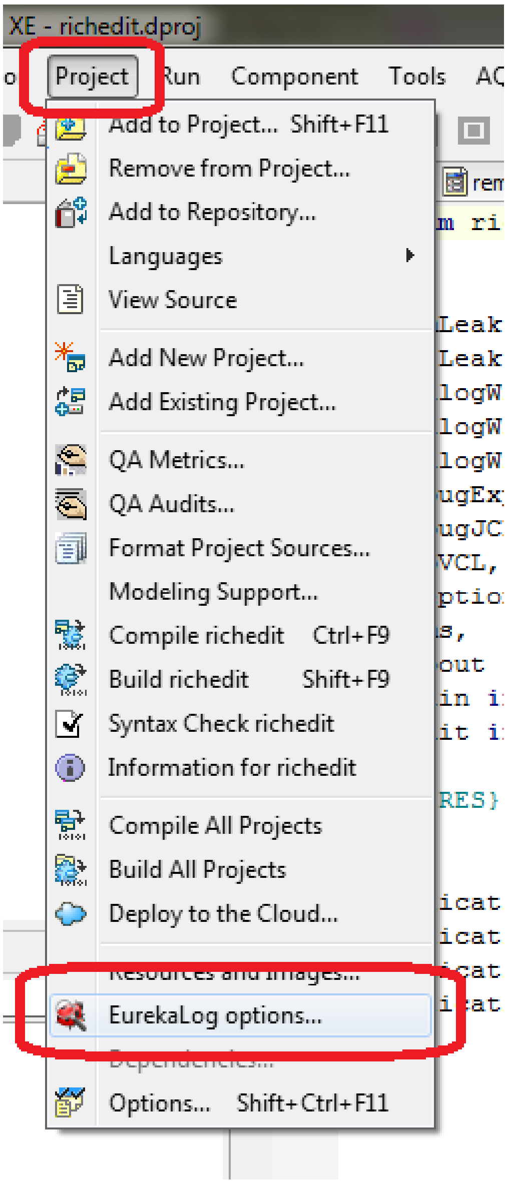
Done!

P.S. You have to install EurekaLog 20 before using it.

Long answer:

First, save your project before enabling EurekaLog.

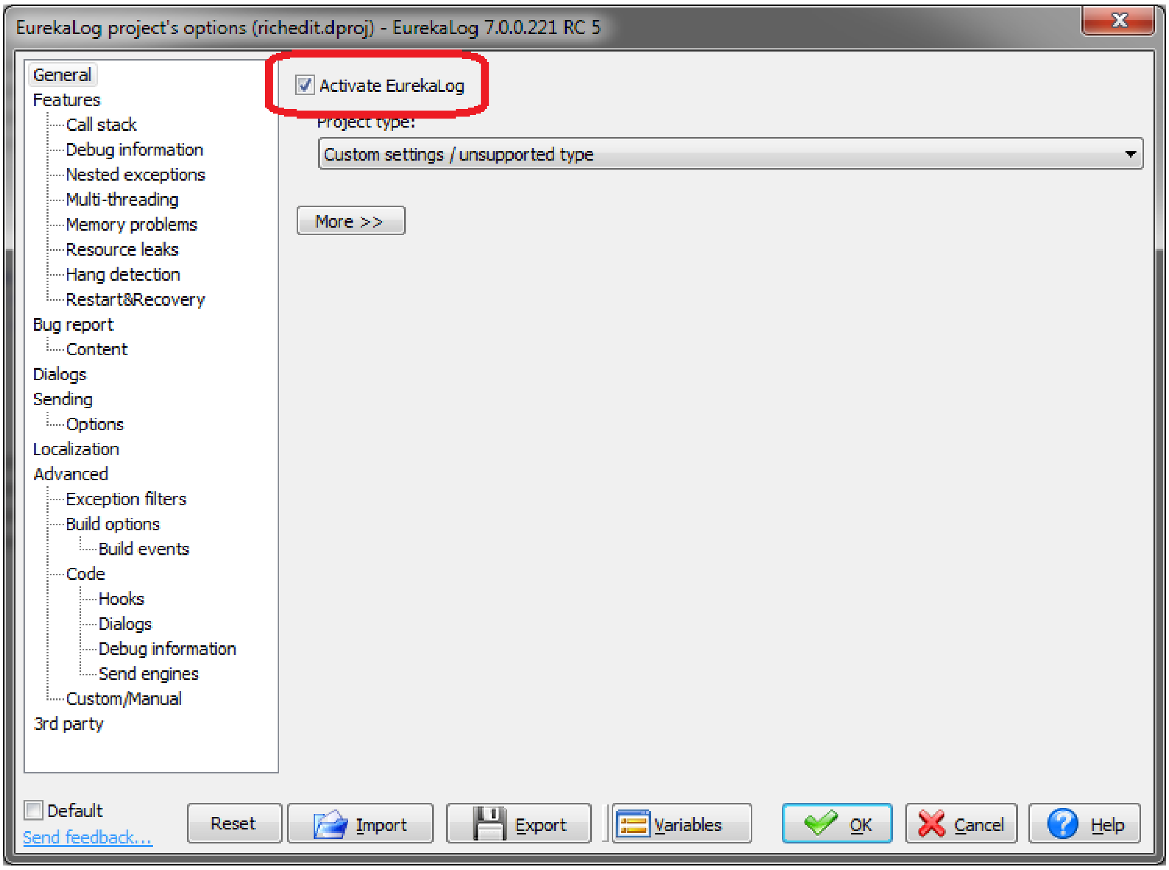
To enable EurekaLog for your application - you need to go to **Project / EurekaLog options** (see also 222):



EurekaLog project options in IDE menu

Don't see EurekaLog menu items? 596

And to enable "Activate EurekaLog" checkbox:

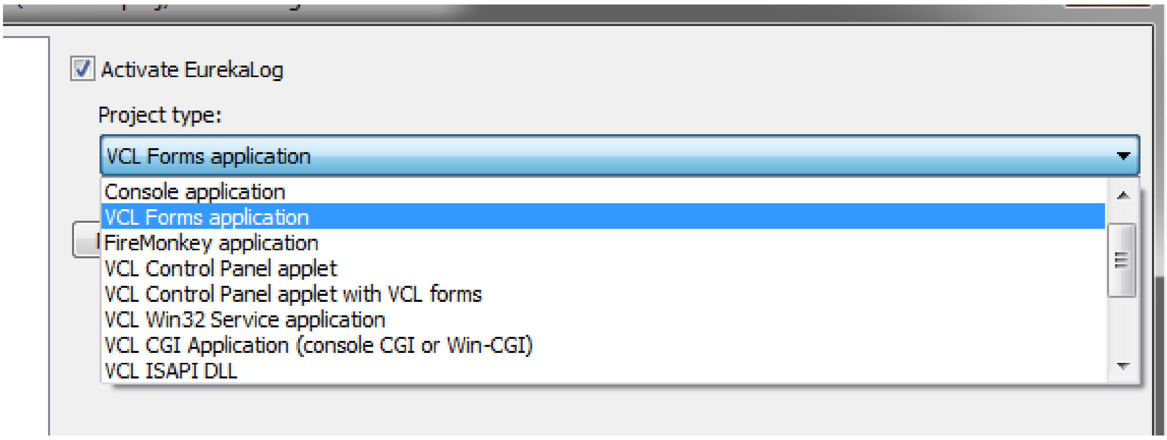


☑ Activate EurekaLog

Project type: Custom settings / unsupported type

General page 234 in EurekaLog project options

You also should select type of your application 363 from drop-down list:


Console application
VCL Forms application
FireMonkey application
VCL Control Panel applet
VCL Control Panel applet with VCL forms
VCL Win32 Service application
VCL CGI Application (console CGI or Win-CGI)
VCL ISAPI DLL

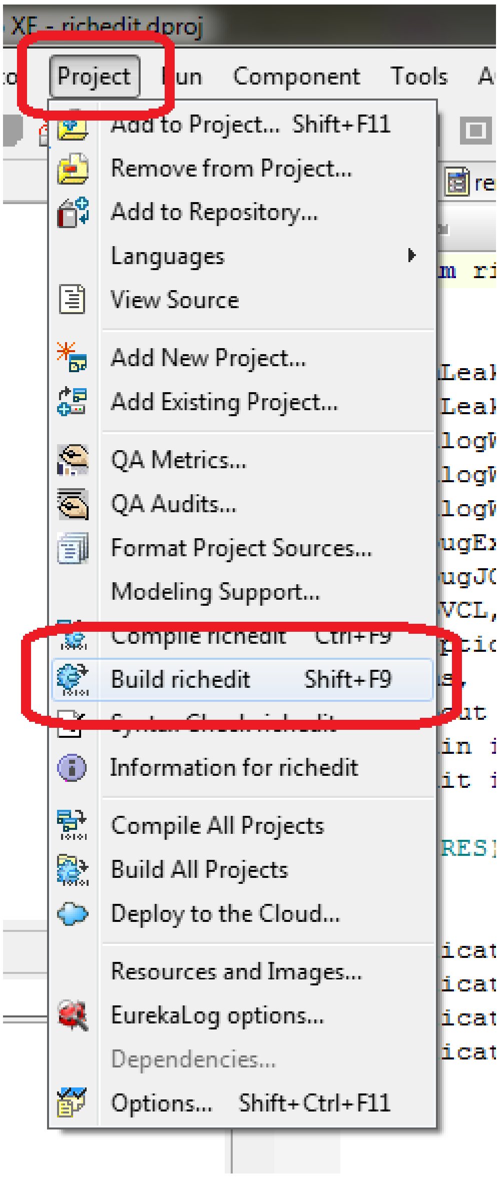
☑ Activate EurekaLog

Project type: VCL Forms application

Selecting type of your application

See supported application types 363 to get more information about each position.

You must recompile your Delphi/C++Builder projects in order to use EurekaLog with them:



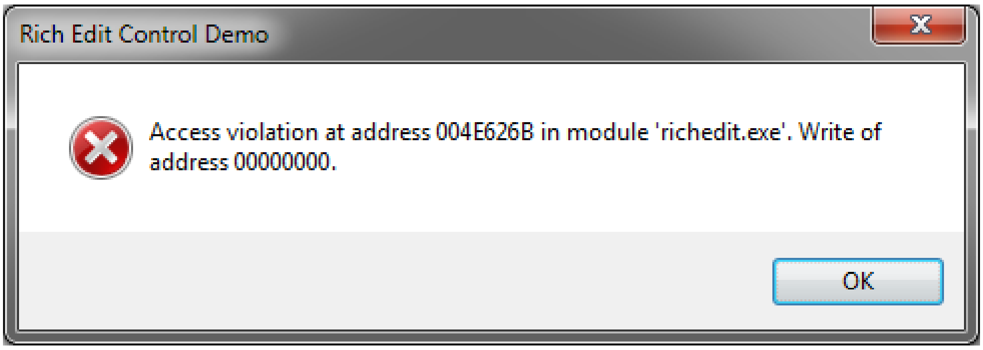
Rebuild your project

EurekaLog will automatically apply all necessary changes to make your projects EurekaLog-enabled.

Done!

Now run your application and observe how your application handles exceptions now:

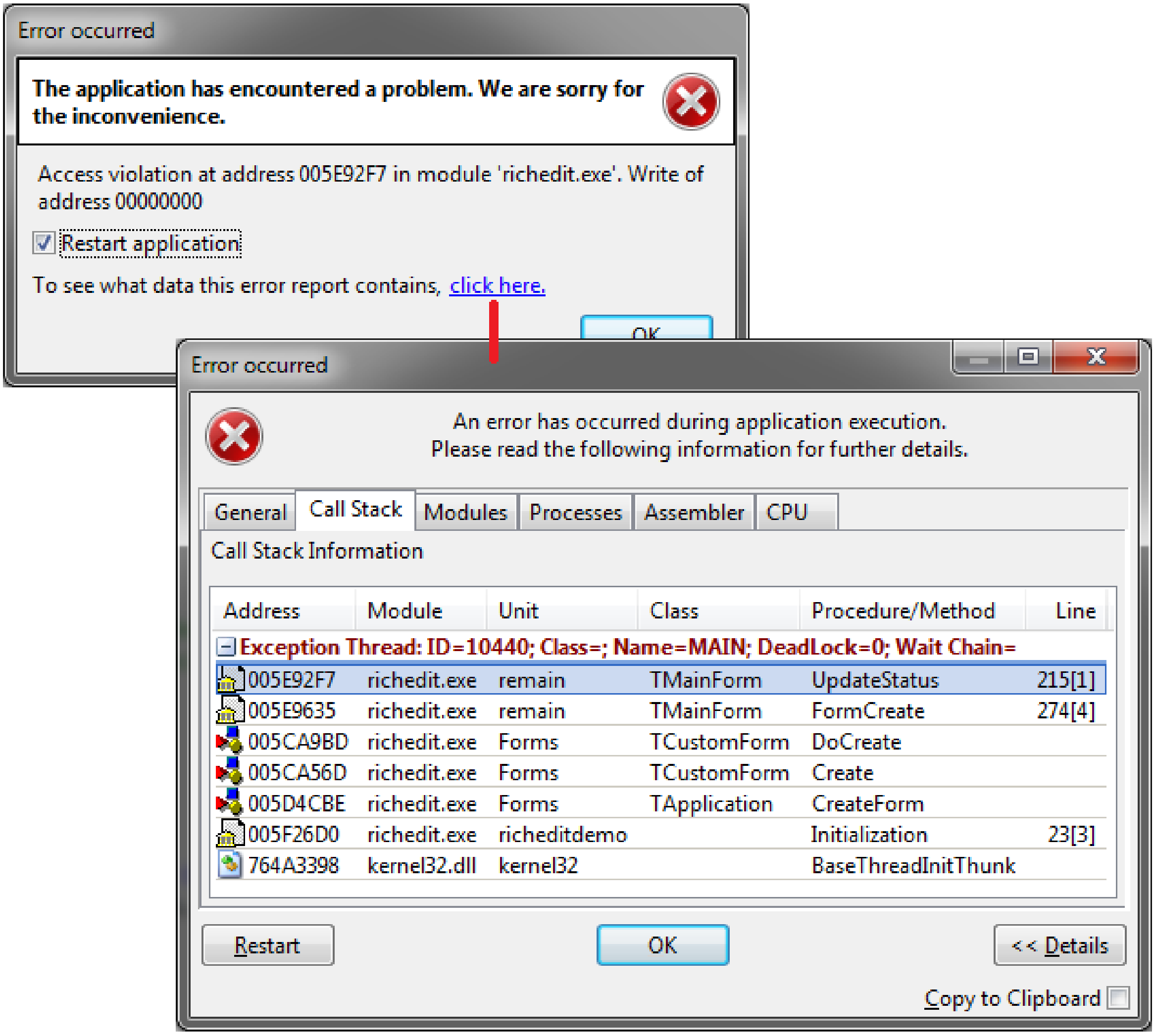
Before



Rich Edit Control Demo

Access violation at address 004E626B in module 'richedit.exe'. Write of address 00000000.

After



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E92F7 in module 'richedit.exe'. Write of address 00000000

☑ Restart application

To see what data this error report contains, click here.

Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=10440; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
005E92F7	richedit.exe	remain	TMainForm	UpdateStatus	215[1]
005E9635	richedit.exe	remain	TMainForm	FormCreate	274[4]
005CA9BD	richedit.exe	Forms	TCustomForm	DoCreate	
005CA56D	richedit.exe	Forms	TCustomForm	Create	
005D4CBE	richedit.exe	Forms	TApplication	CreateForm	
005F26D0	richedit.exe	richeditdemo		Initialization	23[3]
764A3398	kernel32.dll	kernel32		BaseThreadInitThunk	

See also:

- Problems? 606
- What's next?
  - See demos in Start Menu (Start / Programs / EurekaLog 7 / Demos)
  - Reading and understanding bug reports 72
  - Common actions / basic procedures 45
  - FAQ 215
- IDE menu items 222
- Compiling your project with EurekaLog 421

4.4 EurekaLog's basics

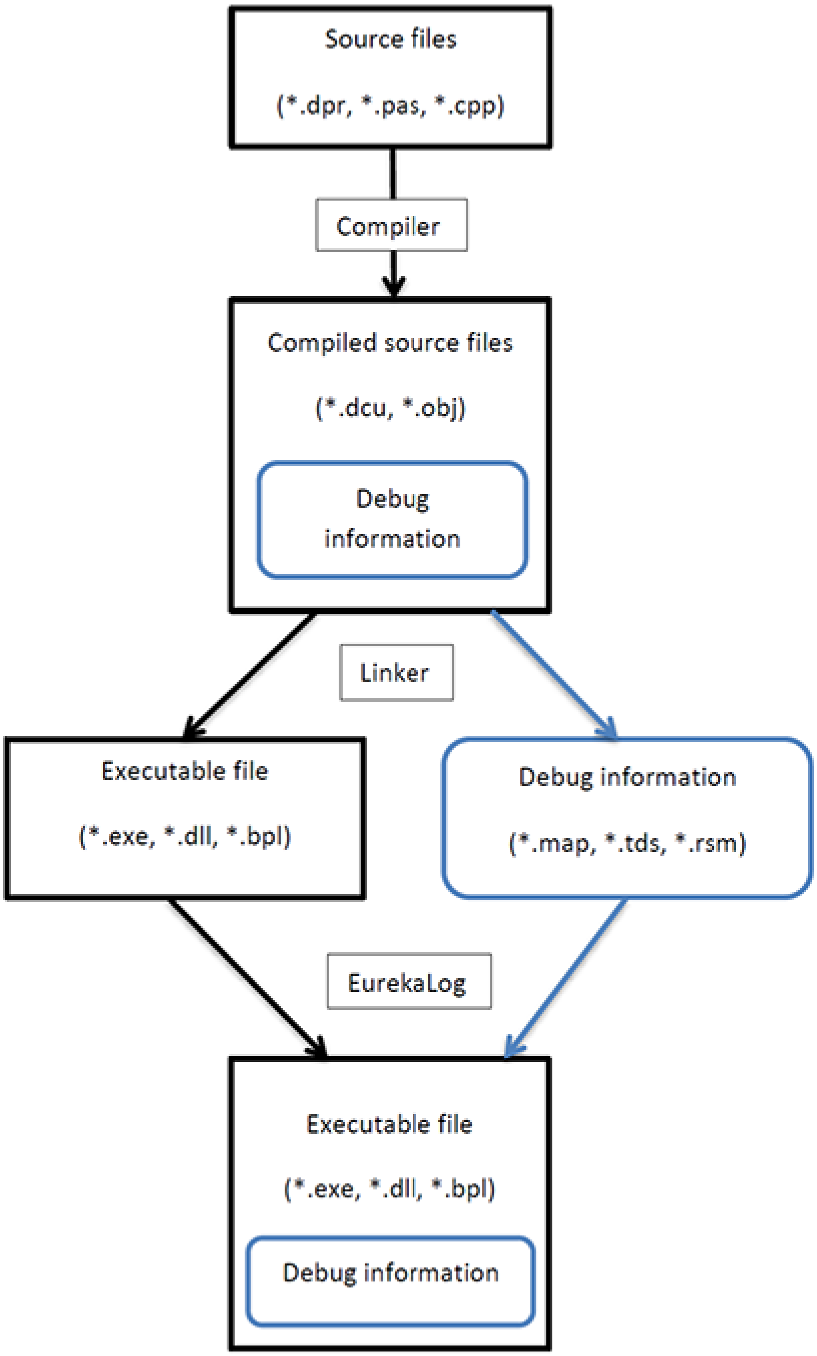
This article is part of Quick start tutorials 20 series.

EurekaLog is injected in your application to work. This includes **EurekaLog's code** and **EurekaLog's data**. Both are equally needed for EurekaLog to work.

- EurekaLog's code is usual Delphi/C++ Builder units, which consists of .pas/.dcu files. Code contains EurekaLog's logic. Most important part of which is exception hooking code. You include EurekaLog code by including EurekaLog's unit into your project (uses clause). Usually this is done automatically when you enable EurekaLog for your project via "Project" / "EurekaLog project options" menu item.

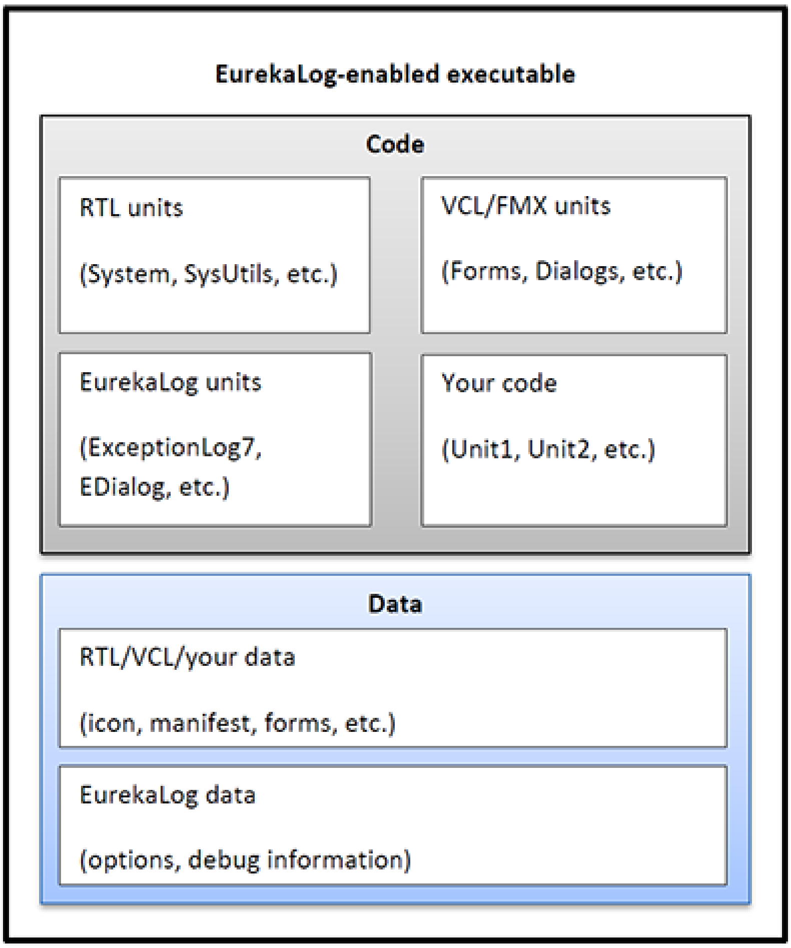
- EurekaLog's data consists of EurekaLog's options for your application and debug information 40. Options are needed to properly setup and configure EurekaLog's behavior on startup. Debug information is needed to build human-readable call stacks in bug reports. EurekaLog's data is injected by post-processing compiled executable with EurekaLog (either by IDE expert or by command-line compiler). Usually this is done automatically when you enable EurekaLog for your project via "Project" / "EurekaLog project options" menu item.

To summarize the whole process:



Build application process with EurekaLog enabled

The resulting executable file will look like this:



Final EurekaLog-enabled executable

You can see EurekaLog code+data.

For more information about compiling your application with EurekaLog see these articles 421.

4.5 Basic terms (definitions/dictionary)

This article is part of Quick start tutorials 20 series.

This is short dictionary of terms used in this help file.

- **Bug** - (see also: **Error**) is a software defect. It is the common term used to describe an *error*, *flaw*, *mistake*, *failure*, or *fault* in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways. Most **Bugs** arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code. A program that contains a large number of **Bugs**, and/or **Bugs** that seriously interfere with its functionality, is said to be buggy. Reports detailing **Bugs** in a program are commonly known as **Bug reports**, fault reports, problem reports, trouble reports, change requests, and so forth. Many people mistakes **Exceptions** for **Bugs** and visa versa, but it's obviously wrong, since **Leak** is an **Bug** too. And not all **Exceptions** are real **Bugs** in your code. For example, you can handle out of disk space exceptions in your code and ask user to free disk. There is no bug in your code.

- **Bug report** - (see also: **Unhandled exception**) a report about possible **Bug** in the application. It usually contains information about **Exception** or **Leak**. **Bug report** usually contains one or more **Call stacks** and some information about system itself and application. **Bug reports** are stored in files. Often one file can hold multiple **Bug reports**. EurekaLog's **Bug reports** have .el, .elp, .elx file extensions. The .el files are usual text files, which can be opened in any text editor. However, EurekaLog Viewer can be used to view them in more convenient way. See also: reading and understanding bug reports 72.

- **Call stack** - stack data structure that stores information about the active subroutines of the application. This kind of stack is also known as an *execution stack*, *control stack*, *run-time stack*, or *machine stack*, and is often shortened to just "*the stack*". **Call stack** is a central piece of information in any **Bug report**. **Call stack** is a sequence of addresses (mostly with textual description - see **Debug information**), which leads us to place of the problem (**Exception** or **Leak**). See also: Call stacks 79.

- **Chained exception** - (see also: **Nested exception**) is **Exception** which was raised inside **Exception handler**.

- **Deadlock** - a situation when two or more threads are waiting for each other infinite. Such cases are **Bugs** in the application. This is a special case of **Hang**.

- **Debug information** - (see also: **Injecting**) is information about link between machine's binary code (.exe/.dll/.bpl file) and source code (.pas file). It's used to build human-readable **Call stacks**. **Debug information** is necessary for EurekaLog to function. It's generated by linker and is saved into .map or .tds files. It's injected into your **Executable** during post-processing stage 421. See also: EurekaLog's basics 38.

- **EDD** - see **Exception Driven Development**.

- **Error** - (see also: **Exception**) see **Bug**. Many people mistakes **Exceptions** for **Errors** and visa versa, but it's obviously wrong, since **Leak** is an **Error** too. And not all **Exceptions** are real **Errors** in your code. Sometimes **Error** can be used as synonym of **Exception**, not **Bug**. It's usually best to avoid this term due to its vague meaning. Use either **Bug** or **Exception**.

- **Exception** - this is an event which interrupts normal execution path of your code and passes control to **Exception handler**. Also, **Exception** is a object which describe exception event. The process of generating **Exception** and interrupting your code is called "raising exception" or "throwing exception". **Exceptions** are raised in case of unexpected or unusual cases. Sometimes (but not always!) **Exceptions** are Errors. For example, "access denied" exception when opening text document in text editor is not a bug in your code, there is nothing to fix. On the other hand, "access violation" exception during the same process of opening is clearly your bug. **Exceptions** can be **Handled** or **Unhandled**. **Exceptions** also can be **Hardware** or **Software**.

- **Exception Driven Development** (EDD) - is a software development process that relies on the capturing software crash and hang data from end-users. You ship your software and analyze incoming bug reports, fix few top bugs and release an update. Repeat iteration again. See also: typical use of EurekaLog 68. This approach is not self-suffice and it can and should be combined with other technics, such as TDD, DDD, XP/Agile, etc.

- **Exception handler** - (see also: **Exception**) is a piece of code which gets control when an **Exception** was raised. It's usually not **Hook**. **Exception handler** either perform cleanup (try-finally) or **Exception handling** (try-except).

- **Exception handling** - (see also: **Exception**) usually refers to the process of "dealing with **Exceptions**". **Exception handling** includes processing of **Exception** such as logging, showing error message, using fallback actions and so forth. **Exception** is destroyed after **Exception handling** is finished. **Exception handling** occurs inside try-except blocks. See also **Handled exception** and **Unhandled exception**.

- **Exception log** - see **Bug report**.

- **Exception tracer** - (see also: **Exception**) a tool which install **Hooks** and intercepts the raising of **Exceptions** and allow user to create **Bug report** for each **Unhandled exception**. Often **Exception tracers** have additional functionality. Like collecting information, debug features, **Leaks** reporting and so on. EurekaLog is an **Exception tracer** tool.

- **Executable** - see **Executable module**.

- **Executable module** - in short, it's .exe, .dll or .bpl file. Basically, it's your compiled project. It's any file that can be executed (can run code).

- **Freeze** - see **Hang**.

- **Handled exception** - (see also: **Exception**) is an **Exception**, which was handled by **Exception handling** process. I.e. this is **Exception** which was processed by **Exception handler**.

- **Hang** - this is situation when application stops responding. This can be a **Bug** in application, but not always. Permanent (endless) **Hang** is always a **Bug**. A special case of **Hang** is **Deadlock**.

- **Hardware exception** - (see also: **Exception**, **Software exception**) is an **Exception**, which is triggered by hardware (like CPU, for example). This type of **Exception** are async **Exceptions** - they can be raised at any place and any time. Access violation (EAccessViolation) is an example of **Hardware exception**. In Delphi: all **Hardware exceptions** are wrapped into exception class delivered from EExternalException.

- **Hook** - a block of code which is called instead or before other code's block. A special case of **Hook** is a simple event/event handler. Usually documented way to call some interceptor code is called "event" and handler is called "event handler". If there is no documented way to intercept code - then it's called **Hooking** and **Hook**.

- **Hooking** - (see also: **Hook**) the process of installing and using **Hooks**. Sometimes it's called **Injecting**, when it specifically refers to undocumented way to intercept code.

- **Injecting** - have different meaning. Can refer to **Hooking** (as the process of installing **Hooks**) or to the process of embedding **Debug information** into **Executable module**.

- **Leak** - is a **Bug** in application when application consumes some kind of resource but is unable to release it back to the operating system. If this resource is memory - then **Leak** is called **Memory leak**.

- **Memory leak** - (see also: **Leak**) is a type of **Leak** when application captures and don't release memory (usually meaning virtual memory).

- **Module** - see **Executable module**.

- **Nested exception** - (see also: **Chained exception**) is the original exception, which has triggered **Chained exception**.

- **Resource leak** - (see also: **Leak**) any **Leak** which is not **Memory leak**. "Resource" can be bitmaps, kernel objects and even other types of "memory".

- **Software exception** - (see also: **Exception**, **Hardware exception**) is an **Exception**, which is triggered by the code. This type of **Exception** are sync **Exceptions** - they can be raised only at specific places, when code has command to raise **Exception**. EStreamError is an example of **Software exception**. This is most common **Exception** type in applications.

- **Unhandled exception** - (see also: **Exception**) may have different meaning. Usually, **Unhandled exception** is an **Exception** which is not **Handled exception**. However, the amounts of "handling" meaning produces different usage cases of "**Unhandled exception**" term. Default meaning is this: **Unhandled exception** escapes any handling code and elevates up to operating system code. Such **Exceptions** means unavoidable death of the process (application) - unless there is some **Hook** for **Unhandled exceptions**. This is a usual meaning of **Unhandled exception** in standard applications. However, when application is enabled by **Exception tracer**, **Unhandled exception** gets second meaning - it now usually means **Exception** that escapes your **Exception handlers** and it's caught by **Exception tracer** (usually via **Hooks**). Event though technically **Exception** is **Handled exception** in this case - it's still can be called **Unhandled exception**. The first case is named "*real unhandled exception*" in this case. It's a very rare case for application with **Exception tracer**.

- **Unit** - Delphi's .pas file or C++ Builder's .cpp file.

Part V

5 Basic procedures

After you've enabled EurekaLog for your application 33 - you probably want to **configure** it, so EurekaLog will better suit your needs.

Usually, our customers do these kind of configurations:
- They set type of their project / application 45.
- They configure bug report 46 (like file storing and content).
- They configure error dialogs 52 (like changing appearance or disabling dialog at all).
- They configure sending report 53 to developer (to receive feedback from field-deployed applications).
- And they adjust the settings of the project itself 58 (to improve detalization of the bug reports).

The best way to ensure satisfaction is to insert an artificial "bug" in your application and ensure that EurekaLog behaves as you expect and want. And if it's not - then adjust EurekaLog's settings 225.

If you write a GUI application - we recommend to place a button on main form and place this code to it's OnClick handler:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  raise Exception.Create('Test exception for EurekaLog');
end;
```

Clicking on such button should invoke EurekaLog's processing and you could check its behaviour on your machine - before deploying application to the clients.

If you configure your application to catch leaks - you can use this code to test EurekaLog's behaviour for leaks:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  AllocMem(10240);
end;
```

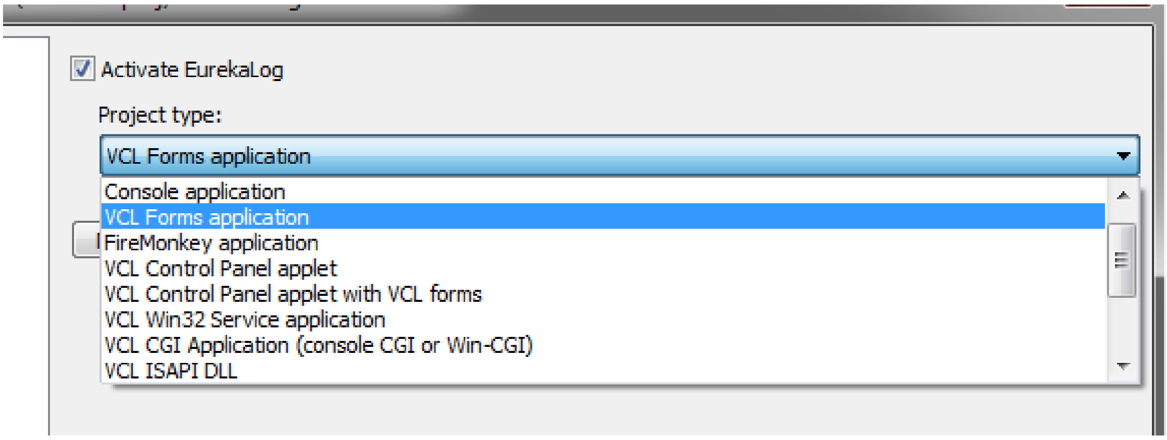
Note: it's important to test both exceptions **AND** leaks, because their processing is very different.

See also:
- Customizing EurekaLog 180
- Video Tutorials

5.2 Selecting application type

This article is a part of basic procedures 45.

Selecting application type 363 is the first action that you should do for your project.



☑ Activate EurekaLog

Project type: VCL Forms application

Console application
VCL Forms application
FireMonkey application
VCL Control Panel applet
VCL Control Panel applet with VCL forms
VCL Win32 Service application
VCL CGI Application (console CGI or Win-CGI)
VCL ISAPI DLL

Selecting a type of your project

You can find this settings on the very first page in EurekaLog project settings: General page 234.

Once you enabled EurekaLog for your project 33, you need to expand the "Project type" combo box, then select a most-appropriated value for your project.

This will changes your project settings for the selected type.

Please, see Application types 363 for more information about each available option.

Now you can set other settings:

- General options 234
- Configuring bug report 46
- Configuring dialogs 52
- Configuring sending 53
- Configuring project 58
- Video Tutorials

5.3 Configuring bug report

This article is a part of basic procedures 45.

When any error occurs (like unhandled exception or leak) - EurekaLog generates a so-called bug report 40. This report can be saved to file and/or send to you (as developer of application). One file can holds many bug reports. See this article 72 for more information about bug reports.

Bug report includes one or more call stacks, some information about application and system, and also may include additional files (like screenshot or current opened document in your application, etc).

Saving bug report to file

You can save bug report to file. Saving bug report is **enabled by default**, so you actually need to disable it, if you don't want/need it. Saving bug report is useful, when you want to analyze the problem after application was closed.

Note: it is recommended to use System Log for Win32 service applications 535.

To enable or disable saving bug report to file - go to "Bug report" section 264 in EurekaLog project options 225 and enable or disable "**Save bug report to file**" option.

Usually EurekaLog bug report file have extensions of .el, .elp or .elx 218 - depending on format of bug report. Only **.el** file extension is available for saving reports - meaning a simple text bug report. Other types of reports (.elx and .elp) are intendant only for sending to developers 53.

You can configure saving bug report to file on the same page.

First, you can specify a **folder to store bug report**. Default is a subfolder in "%APPDATA%\Neos Eureka S.r.l\Bug reports\" (see also 217). Empty string means default path (i.e. the same folder - "%APPDATA%\Neos Eureka S.r.l\Bug reports\").

Note: you can open this folder with bug reports by opening **Start** / **Programs** / **EurekaLog** / **Bug reports** menu item.

If you want to save bug report to the same folder as executable module - use ".\" folder. You can also use any other relative path, like this for example: ".\Reports".

Note: if your selected folder will be write-protected at run-time, EurekaLog will revert it to default. If path doesn't exist - it will be created.

All bug reports are saved in the same file, so new bug reports do not overwrite old reports - they appended to the end of the same file. You can specify a **maximum amount of reports**, which one file can hold. By default it's 32 (it's "Max reports in one file" option). A typical default bug report has 100 Kb in size, so 32 reports will take approximately 3,2 Mb at maximum. When limit is reached (by count, not by size), the oldest report (which is stored first) will be deleted.

If you don't need or want to store multiple reports - you can specify a value of 1. This will emulate case "new report overwrites old". This behaviour is a good idea for developer's machine.

Since capacity of file can be limited, you may be interested in storing information with maximal usefulness. For example, you may don't want for this file to hold a 32 bug reports of the very same error, which occurs 5 times a day. You can enable "**Do not save duplicate errors**" option to take care of this situation. When this option is enabled - file will contain only unique bug reports. If current bug reports represents an problem, which was already saved in the file - the file will not be appended with new bug report. Instead, a error's count will be increased as +1. How EurekaLog knows, which reports are duplicates and which are not? It uses a so-called Bug ID 421. Reports with the same Bug ID's values are treated as equal (even though they aren't - obviously, their date-times are different). Reports with different Bug IDs will be saved to file anyway.

Option "**Delete file at version change**" may be useful, if you don't want to get old bug reports, when you've released a new version of your application. If you enable this option - then be sure to use version information in your project and to increase it with new build/release. Once enabled, this option will delete bug report's file, if it holds reports from previous version of your application.

Configuring bug report's content

Call stack is a central piece of information inside any bug report. Since it plays such important role - there are many options and features for call stacks. Because there is a lot of information to describe these options - this information is separated into its own article 48.

You can configure other information (not call stack; auxiliary information) which you want to include in the bug report. Just go to "Bug report / Content" section 266 in EurekaLog project options 225 and enable or disable parts of information to include in bug report.

While some people tends to enable *everything* - it may be not be a very good idea. Collecting information takes time. Collecting everything will result in freezing your error dialogs for short amount of time. Is it really needed? If your application doesn't work with printer - why collect printer information? If your application doesn't communicate with other applications - what can be useful in processes list?

Usually, it's a lot better to include only required information - disable everything and enable only option which can be useful for your kind of application. For example, a Win32 service can collect information about current account and its privileges. A text editor with printing feature can capture information about installed printer. An application with extensive graphic work can include a monitor and video-card information. And so on.

Also, be aware that your application may be considered as harmful (spy-ware) by customers, if it collects extensive information which is definitely not needed (like monitor information for non-GUI applications).

Note: project options may affect bug report detalization 58.

See also:

- Configuring call stack 48
- Bug report options 264
- Bug reports 72
- System Logging setup 535
- Configuring dialogs 52
- Configuring sending 53
- Configuring project 58
- Customizing EurekaLog 180
- Video Tutorials

5.3.1 Configuring call stack

Call stack is a central piece of information in any bug report. Call stack is a stack data structure that stores information about the active subroutines of the application. Call stack is a sequence of addresses (mostly with textual description), which leads us to place of the problem. This kind of stack is also known as an *execution stack*, *control stack*, *run-time stack*, or *machine stack*, and is often shortened to just "*the stack*".

Two main sources of information for call stack:
1. Stack tracing method (run-time);
2. Debug information (design-time).

Stack tracing method collects return addresses at run-time and defines list of entries in call stack. Debug information establishes correspondence between RAW addresses and source code. It is generated at design-time. It defines human-readable form of call stack:

1. Executable's code -> Exception -> Stack Tracing Method -> list of RAW addresses;
2. List of RAW addresses -> Debug Information Provider + Debug Information (executable's data) -> human-readable call stack.

Stack tracing methods
Stack tracing method scans CPU stack and retrieves all found return addresses. The result will be a list of RAW pointers. Each pointer in the list is a return address - i.e. it points to some location in code section. Stack tracing method determinates list of addresses. Different tracing methods may produce different lists.

To learn more about different stack tracing methods - see this article 578.

Stack tracing method can be configured here 237.

Debug information
Debug information is information about link between machine's binary code (.exe/.dll/.bpl file) and source code (.pas file). It is used to build human-readable call stacks. Debug information is necessary for EurekaLog to function. It is generated by linker and is saved into .map or .tds files. It is injected into your executable during post-processing stage 421. Debug information can be generated by various tools (compilers/linkers), and in different formats. EurekaLog must know debug information format in order to use debug information in that format. Code which knows some debug information format and is able to read debug information in this format is called "Debug Information Provider".

EurekaLog supports the following debug information providers 409. Debug information can also be converted from one format to another by using special tools 516.

Debug information providers can be configured here 355.

Note: debug information itself is not configured in EurekaLog's options. It must be configured via compiler/linker/converter's options. See also: configuring project for EurekaLog 58. Or it can be supplied by debug information converters 516. See also: using EurekaLog with 3rd party tools 514.

Other important considerations for call stacks

Call stacks for chained exceptions
Chained exception is an exception which occurs during processing of another exception 40. That "another" (original) exception is called "nested exception". For example:

```
try
  // Low-level error (a.k.a.  original, first, bottom, inner, nested)
  raise ERangeError.Create('Invalid item index');
except
  // High-level error (a.k.a. introduced, last, top, outer, chained)
  raise EFileLoadError.CreateFmt('Error loading file %s', [FileName]);
end;
```

As you can see, low-level exception (nested) is the exception you're interested in. It indicates a reason for failure. This is what you typically want to be logged. Chained exception is triggered by original exception and provides more descriptive error message. So, you typically want to show it to user as error message.

Thus, typically you want first exception to be logged, but last exception to be shown to end user. Classic/default Delphi and C++ Builder behavior is to work only with last exception always. Default settings for EurekaLog is to log original (nested) exception, but show chained exception to user.

To learn more about chained/nested exceptions support in Delphi/C++ Builder and EurekaLog - see this article 573.

Chained exceptions can be configured here 244.

Important note: this feature require EurekaLog to be able to track life-time of exception objects. Therefore, it's highly recommended that you enable the following options:

- "Enable extended memory manager" option 250
- "Use low-level hooks" option 259
- "Capture stack only for exceptions from current module" option 237

Otherwise it's recommended that you keep all options on this page into "Classic" position, or EurekaLog *may* show information about wrong exceptions. Exceptions passed between executable modules (i.e. from/to DLLs) will probably won't be able to work properly with this feature. However, exceptions that are converted to error code/HRESULT/etc. when passed between modules boundaries will work OK.

Multiple call stacks in a single bug report / Bug reports for multi-threaded applications
A single bug report may contain more than one call stack. Additional call stacks may be created for leaks or background threads.

See this article 166 to learn more about leak reports. See this article 547 to learn more about multi-threaded applications.

Deferred and immediate call stacks
EurekaLog must collect information about exception when this exception is raised (information such as call stack, time, etc.). Collecting this information will take some time. This may become a performance issue if your application raises exceptions too often - and handles them immediately (i.e. it uses exceptions as part of its normal execution path). This would mean that even though information for exceptions is collected, but it is not used - since bug report is not created (because application handles exceptions by itself). Then enabling EurekaLog will introduce slowdown for such application.

We recommend to review your code and avoid raising exceptions too often (i.e. avoid using exception as part of normal execution path; use exceptions only for errors/rare conditions).

A typical fixes for your code include:

- Avoid raising exception when you can pre-check error-condition. For example, it's better to use TryStrToInt or StrToIntDef instead of StrToInt + try/except block;
- Do not raise exception for non-errors. If you still need to do this - consider creating custom exception class for this purpose and exclude exception from EurekaLog (see next item);
- You can create custom exception classes for your purposes. You can mark some exception classes as "ignored" for EurekaLog. You can do this via filters 185, events 192 (specifically: OnRaise), attributes 190, or low-level handlers 211;
- You can also add `SetEurekaLogStateInThread(0, False)` and `SetEurekaLogStateInThread(0, True)` around blocks of code which can raise exceptions intensively, but your code handles all these exceptions.

However, x86-32 platform has unique architecture: a call stack may be build later - during exception handling step. This feature can give a performance gain, because exception tracer now may build call stack later - at handling state, it's not strictly necessary to build call stack when exception is raised. Thus, there will be a huge performance boost, if most of raised exceptions are handled by your code and not reach default handler (which will create a bug report and, thus, a call stack).

This feature is enabled by default, but it is only applicable to Win32 platform.

See this article 583 to learn more about deferred and immediate call stacks.

Exceptions in DLLs

The default good practice when working with exceptions and DLLs is not to let any exception escape DLL 455. That's because DLL and exe can be written in different programming languages, and caller may not know how to handle (and release) exception object from callee. That's why by default EurekaLog is configured to handle only exceptions inside current executable.

It's highly recommended to follow best practices 457. However, you still may want to handle DLL's exceptions in exe (or visa versa). For example, if you're 100% sure that both DLL and exe is written in the same programming language (and is compiled by the same compiler's version). Then you can instruct EurekaLog to catch exceptions from other modules. You can enable this behavior by disabling "Capture stack only for exceptions from current module" option 237.

Note: you should probably disable chained exceptions support 573 for DLLs that let exceptions escape DLL and be handled by the caller (see above). This feature requires ability to track life time of exceptions objects. This is not possible for general case (e.g. host and DLL are compiled by different compilers and there is no assist from RTL for tracking exception objects). This feature *may* work in some specific configurations.

See also FAQ below to learn more about building call stacks for DLLs.

How to...

...show RTL and VCL units in call stack?

Set "Detalization level" option 237 to "Show any (including RAW addresses)" or "Show items with procedure name (DLLs)".

...show RTL and VCL units with line numbers in call stack?

Enable "Use Debug DCUs" option and build the project (make/compile is not enough).

...hide RTL and VCL units in call stack?

Disable "Use Debug DCUs" option, set "Detalization level" option 237 to "Show items with unit name (BPLs)" or "Show only items with full info (line number available)", and build the project (make/compile is not enough).

...show RTL and VCL units in call stack for packaged applications?

Enable "JEDI (JclDebug)" option 355. Distribute *.jdbg files (along with *.bpl files) with your application.

Notes:

- *.jdbg files can be found in \bin folder of your IDE installation;
- *.jdbg files are not available for very old Delphi/C++ Builder versions.

...show JCL/JVCL units in call stack for packaged applications?
Enable "JEDI (JclDebug)" option 355. Enable "Packages / Create MAP files / Create JEDI Debug Informations" and "Packages / Create MAP files / Insert JEDI Debug Informations into the libraries" options during installation of JCL/JVCL.

...capture exceptions from DLLs?
Disable "Capture stack only for exceptions from current module" option 237. Enable "DLL export table" option 355 **or** provide debug information for DLL by compiling DLL project with EurekaLog ("DLL" profile 368).

Normally this practice is not recommended. Recommended practice is to handle exceptions within the same module and pass error condition (flag, error code, HRESULT, etc.) to the caller 457. See also: using EurekaLog with DLLs 455.

...how to work with call stacks in COM application?
See this article 488.

...show system DLL functions in call stack?
Enable "DLL export table" option 355.

...show system DLL functions (including internal functions) in call stack?
See this article 504.

...hide system DLL functions in call stack?
Disable "DLL export table" option 355, disable "Microsoft DBG/PDB" option 355, and set "Detalization level" option 237 to any value except "Show any (including RAW addresses)" and "".

...show custom DLL/package functions in call stack?
Enable "DLL export table" option 355.

Note: this question is independent from "How to capture exceptions from DLLs" question above.

See also this article 495.

...show custom DLL/package functions with line numbers in call stack?
Compile your DLL or package project with EurekaLog (use "DLL" profile 368 for DLL and "Package" profile 370 for packages).

Note: "Standalone DLL" profile 369 can also be used for DLL, but it is designed for different usage case 480 (such as COM, plugins or any other usage of EurekaLog-enabled DLL inside non-EurekaLog-enabled host).

...show custom DLL functions with line numbers in call stack (DLL is compiled by non-Embarcadero compiler)?
a). [Microsoft Visual Studio only] Enable "Microsoft DBG/PDB" option 355, set "Debug Information Format" option in your Visual Studio DLL project to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler), enable "Generate debug info" option in your Visual Studio DLL project (/DEBUG option for linker). Distribute generated .pdb file with your .dll file. **Note:** it is not possible to create an .exe or .dll that contains debug information. Debug information is always placed in a .pdb file.

or:

b). Enable "Microsoft Dbg/PDB" option 355, set "Debug Information Format" option in your Visual Studio DLL project to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler), enable "Generate debug info" option in your Visual Studio DLL project (/DEBUG option for linker). Post-process your DLL file with EurekaLog command-line compiler 426. Use NUL as project file name for --el_alter_exe switch and don't forget to add --el_source=PDB switch. You also have to create .eof file for your DLL and specify path to it via --el_config option. **Note:** most options in .eof file will be ignored. Only design-time options will have effect (such as password encryption for debug information, etc.).

See this article 496 for more details.

...minimize information in debug information/call stack (for shareware applications)?
Enable "Do not store class/procedure names" option 243.

See this article 585 for more information.

...hide certain routines from call stack?
Wrap such routines in `{$D-} { routine code here } {$D+}`.

See this article 585 for more information.

See also:
- Configuring bug report 46
- Using EurekaLog in multi-threaded applications 547
- EurekaLog's basics 38
- Configuring project for EurekaLog 58
- Compiling with EurekaLog 421
- Using EurekaLog with exceptions in DLLs 455
- Multi-threaded call stacks 85
- Stack tracing: RAW method and frame-based method 578
- Stack tracing: deferred vs. immediate 583
- Nested/chained exceptions 573
- Using Microsoft DbgHelp DLL 504
- Debug information settings 243
- Using debug information converters 516
- EurekaLog for shareware developers 585

5.4 Configuring dialogs

This article is a part of basic procedures 45.

When any error occurs (like unhandled exception or leak) - EurekaLog generates a bug report, which you configured on previous step 46. EurekaLog also shows an **error dialog** - it's some kind of dialog or output, which tells the user about error.

You can configure dialogs at "Dialogs" section 267 in EurekaLog project's options 225.

Usually a dialog is set automatically when you select a type of your application 45. In some cases you have only single meaningful type of dialog available. But in most cases you can change a default dialog to some other dialog.

Just select a dialog's type from combo box and set additional dialog display options.

There are many dialogs available - here is a short list of available dialogs:

Dialog	Where applicable	Description
(none) 371	Any application's type	Dialog that does nothing at all. It doesn't show anything.
RTL 371	Any application's type	Standard application's error dialog.
MessageBox 373	Visual applications only	Displays error message via Windows.MessageBox function.
MS Classic 377	Visual applications only	Displays error message in MS Windows XP-style dialog.

EurekaLog 379	Visual applications only	Displays error message in EurekaLog-style dialog.
Console 382	Console applications only	Displays error by outputting it to console (error output).
System log reporting 384	Any application's type	Outputs error message to system log 535 (event log).
WEB 386	Web applications only	Displays error in returned HTML page.
Windows Error Reporting 389	Any application's type	Elevates error to OS.

Note: some dialogs have meaning (and can work) only in certain application types. For example, attempt to use console dialog in VCL GUI application will result in error. Other dialogs may require additional setup. For example, a system logging requires you to register an event source for your application. See dialogs 370 for more information.

You can know more about each dialog's options by reading Dialogs 370 descriptions or by clicking on the dialog's link in the table above.

See also:

- Dialogs options 267
- Configuring bug report 46
- System Logging setup 535
- Configuring sending 53
- Configuring project 58
- Customizing EurekaLog 180
- Video Tutorials

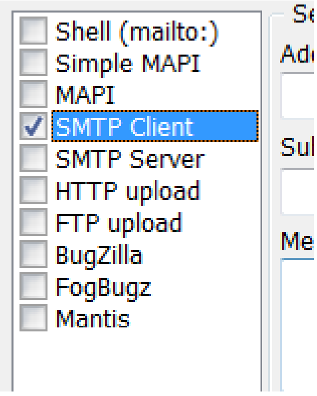
5.5 Configuring sending report

This article is a part of basic procedures 45.

If you're waiting for your clients to tell you about problems in your application - then you see only a tiny fraction of all problems with your application 68. That's why you need to build an exception and error reporting facility. And EurekaLog is able to help you with that task: it can send a bug report about each problem in your application, deployed on clients' machines. Reports can be send automatically, silently or with client's approval. You have a bunch of send methods available.

You can configure this behaviour at "Sending" section 302 in EurekaLog project's options 225.

You should select one or more sending methods - by checking check-box on the left:



☐ Shell (mailto:)
☐ Simple MAPI
☐ MAPI
☑ SMTP Client
☐ SMTP Server
☐ HTTP upload
☐ FTP upload
☐ BugZilla
☐ FogBugz
☐ Mantis

One sending method selected

You need also to setup selected method in the right part. Each method has its own unique options. Usually, you just need to fill details about your account (like login / password).

All default build-in methods can be divided into two categories:
1. E-mail based. These methods send bug report inside e-mail message to your e-mail address.
2. Web based. These methods uploads bug report via HTTP or FTP protocols (including bug-trackers with web interface).

- Shell, Simple MAPI, MAPI, SMTP Client and SMTP Server are e-mail based send methods.
- HTTP upload, FTP upload, BugZilla, FogBugz, Mantis and JIRA are web based send methods.
- BugZilla, FogBugz, Mantis and JIRA are web-trackers.

Unfortunately, there is no best sending method - each way have its own advantages and drawbacks. Please, see Selecting send method 55 to pick a method which matches your needs.

Note: you can select more than one send method and change methods order. If first method will fail sending, the second selected method will be used. If second method will fail too, the third method will be used. And so on, until send will success or there will be no send methods left. The sending phase is considered to be "OK", if one method was able to send report. If all methods had failed - then sending phase will be considered as "failed".

Common "gotcha's" for sending bug reports

Here are some points which are worth looking for:
- Try to avoid non-ASCII characters (those which code is above 128) in host/URLs, account names, passwords, etc. While most recent environments offer full support for localized names and characters, older platforms may limit EurekaLog capability to use them. For example, using Cyrillic account name in Delphi 7 will break sending on Italian Windows - because ANSI string will be treated in wrong code page.
- Use several send methods for best delivery results. See Selecting send method 55 for more info. It's best to use one method of your choice and back it up with one of few e-mail sending methods. Thus, if one method fails - another one will succeed. If you use web-tracker as your primary method - often you can configure it to parse e-mails too 153.
- Create a new account specifically for sending reports. **NEVER** use main/personal/administrator account for bug report submission. Create a new e-mail account (if it's possible - limit its rights to send only). Create a new FTP account (limit its rights to upload files to specified folder only). Create new web-tracker account (limit its rights to submitting reports only).
- Create a new "project" or account for each of your products. Do not mix several products with one account. For example, create different "projects" in bug-tracker software for each of your software products.
- Test sending before deploying. Test it with both exceptions and leaks. Test it for new and closed reports. Be sure that sending process meets your expectations.
- Before upgrading/changing your end of bug report submission (HTTP upload script, FTP configuration, bug tracker software, etc) - be sure to test this new environment. Ensure that new configuration allows old versions of your application to report bugs (if you still need these bug reports). For this reason be extra careful to use "hosted" solution - because you may not control server software changes.
- There are events for customizing sending: such as OnAttachedFilesRequest, OnZippedFilesRequest, and OnCustomWebFieldRequest.

What's next?

After you start receiving bug reports "from the field" - it's time to read them 72 and solve the bugs 72.

See also:
- Sending options 302
- Managing bug reports in issue tracker 105
- Security Considerations 158
- Configuring bug report 46

- Configuring dialogs 52
- Configuring project 58
- Customizing EurekaLog 180
- Video Tutorials
- Customizing EurekaLog 180

5.5.1 Selecting send method

Unfortunately, there is no best sending method for your bug reports 53 - each way have its own advantages and drawbacks.

Important note: there are events for customizing sending: such as OnAttachedFilesRequest, OnZippedFilesRequest, and OnCustomWebFieldRequest.

First, let's start with common properties for each group:

E-mail methods

Advantages:
- Simple and familiar.
- Can be send manually (good as "last resort measure").
- Clients most likely have e-mail access, so sending have good chances to succeed.
- Good for basic support for unsupported web-trackers (see also 153).

Drawbacks:
- Depends on client's environment. You can't control it.
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management (however, you can use EurekaLog Viewer 617 to collect reports from e-mail account).

Web methods

Advantages:
- Simple and controllable.
- May be customizable.
- Simple setup.
- SSL/TLS support.

Drawbacks:
- May be blocked by firewall.
- No bug report management.
- User e-mail address is optional.
- Require hosting or server.

Web-trackers methods

Advantages:
- Powerfull and customizable.
- Bug report management.
- SSL/TLS support.

Drawbacks:
- May be blocked by firewall.
- Requires setup.
- Require hosting or server + database.
- User e-mail address is optional.
- Requires setup for each your project.

Note: bug trackers and e-mail methods are close-related: bug trackers are usually able to parse incoming e-mails and insert them as issues into database. Thus, you can get non-supported bug tracker working 153 (by setuping it to parse e-mails and setup e-mail sending in EurekaLog). Also, web trackers usually can generate e-mail notifications about new issues.

Now, let's take a look at each method (common advantages/drawbacks aren't listed). Only major points are listed below. For detailed pros/cons of each method - please see description of each method 390.

E-mail: Shell 391 (mailto protocol)

Advantages:
- Available always

Drawbacks:
- High amount of limitations (not customizable, no attaches, etc.)

Conclusion: good for "last resort" measure, but have **many** limitations.

Note: due to "no way to get real send result" - it's best to place this method last, if you select several methods.

E-mail: Simple MAPI 393

Advantages:
- Most common protocol for 3rd party e-mail clients

Drawbacks:
- Obsolete protocol, not supported by modern Outlook

Conclusion: good for e-mail sending with using client configuration.

E-mail: MAPI 396 (also known as "Extended MAPI" or "MAPI 1.0")

Advantages:
- Supported by Outlook and Exchange

Drawbacks:
- Not supported by other e-mail clients

Conclusion: good as alternative for SMAPI in case your clients use Outlook/Exchange.

E-mail: SMTP client 397

Advantages:
- Most reliable e-mail protocol

Drawbacks:
- You must store your real e-mail account details (login/password) in your application

Conclusion: only good if other methods are blocked. But if you can afford storing password - then it will be better than SMTP server.

E-mail: SMTP server 398

Advantages:
- Most powerful method with low limitations

Drawbacks:
- It's usually blocked by ISPs to block spam software

Conclusion: good method with low limitations, but only if it's not blocked by client's firewall or ISP.

Web: HTTP 398

Advantages:
- Simple and highly customizable.

Drawbacks:
- A lot of custom work to get anything above simple upload functionality.

Conclusion: good as base for building custom sending. For simple file uploads - FTP is a preferred way.

Web: FTP 404

Advantages:

- Minimum efforts to setup (among all web methods, including web-trackers), very reliable

Drawbacks:

- No other functionality except simple bug reports upload

Conclusion: good if you need simple and reliable file send (upload). Often it is better than e-mail methods.

Web-trackers

No additional points, except common items listed above in the "**Web-trackers methods**" section. Please read through descriptions of each bug tracker 390. There are also setup manuals 105 available. You can read through manual to get idea of bug tracker's capabilities and working process.

See also:

- Detailed description of each send method 390.
- Comparison of issue-tracking systems.

Conclusion: best if you need a complete bug tracking solution. For simple file sending HTTP or FTP is preferable.

Remarks:
(*) That's because, if you have two e-mail client installed (say, Windows Mail and Outlook) - both will definitely support mailto protocol, but only one can support simple MAPI, so you may launch non-default e-mail client (which is not configured). For example, if you have Outlook 2010 as your default e-mail client and you use simple MAPI - it will launch Windows Mail client, because Outlook 2010 doesn't support simple MAPI. The same example holds true for MAPI, if you use Windows Mail as your default e-mail client (because Windows Mail doesn't support MAPI).

Recommended order of send methods

1. The method of your choice (like HTTP or Web-tracker)
2. SMTP Server or SMTP Client (depending on whenever can you store your account details in application or not)
3. MAPI or Simple MAPI or both
4. Shell (mailto)

If you don't want (or can't) use some method in this list - just exclude it and place the rest in this order. For example, if you want "mail only" delivery - use this:

1. SMTP Server
2. MAPI
3. Simple MAPI
4. Shell

This is just recommendation, not final rule. For example, you may use this sequence:

1. Mantis.
2. Shell.

It's your choice, it's up to you.

Note: usually there is no big reason to enable both SMTP client and SMTP server modes. Use either first or second, but not both. I.e. if you can afford storing the password from your real e-mail account in application - use SMTP client. Otherwise use SMTP server.

See also:
- Configuring sending 53
- Description of available send methods 390
- Using unsupported bug tracker software 153
- Managing bug reports in issue tracker 105
- Security Considerations 158
- Customizing EurekaLog 180

5.6 Configuring project itself

This article is a part of basic procedures 45.

You can improve detalization of your bug reports by selecting the proper project options. This articles explain various project options related to debugging and a recommended choices.

Short answer:

1. To solve problems with non-working, partial or misleading call stack - clear .map, .tds and .dcu files of your project and be sure to:

- (Delphi) enable "**Compiler"/"Debug information**", "**Linker"/"Map file**" = "Detailed".
- (C++ Builder) enable "**C++ Compiler"/"Debugging"/"Debug information**", "**C++ Compiler"/"Debugging"/"Debug line number information**", "**C++ Linker"/"Full debug information**", "**C++ Linker"/"Output"/"Map file**" = "Detailed segment map".

2. To increase help level of EurekaLog:

- (Delphi) enable "**Compiler"/"Stack frames**", "**Compiler"/"Range checking**" and "**Compiler"/"Use Debug DCUs**" options and make "**Project"/"Build all**".
- (C++ Builder) disable "**C++ Linker"/"Output"/"Map with mangled names**" option, enable CodeGuard for you application (some options may conflict with EurekaLog).

Note: exact options names and locations depend on your IDE version. The specific option may be called differently or be located in other place in your IDE.

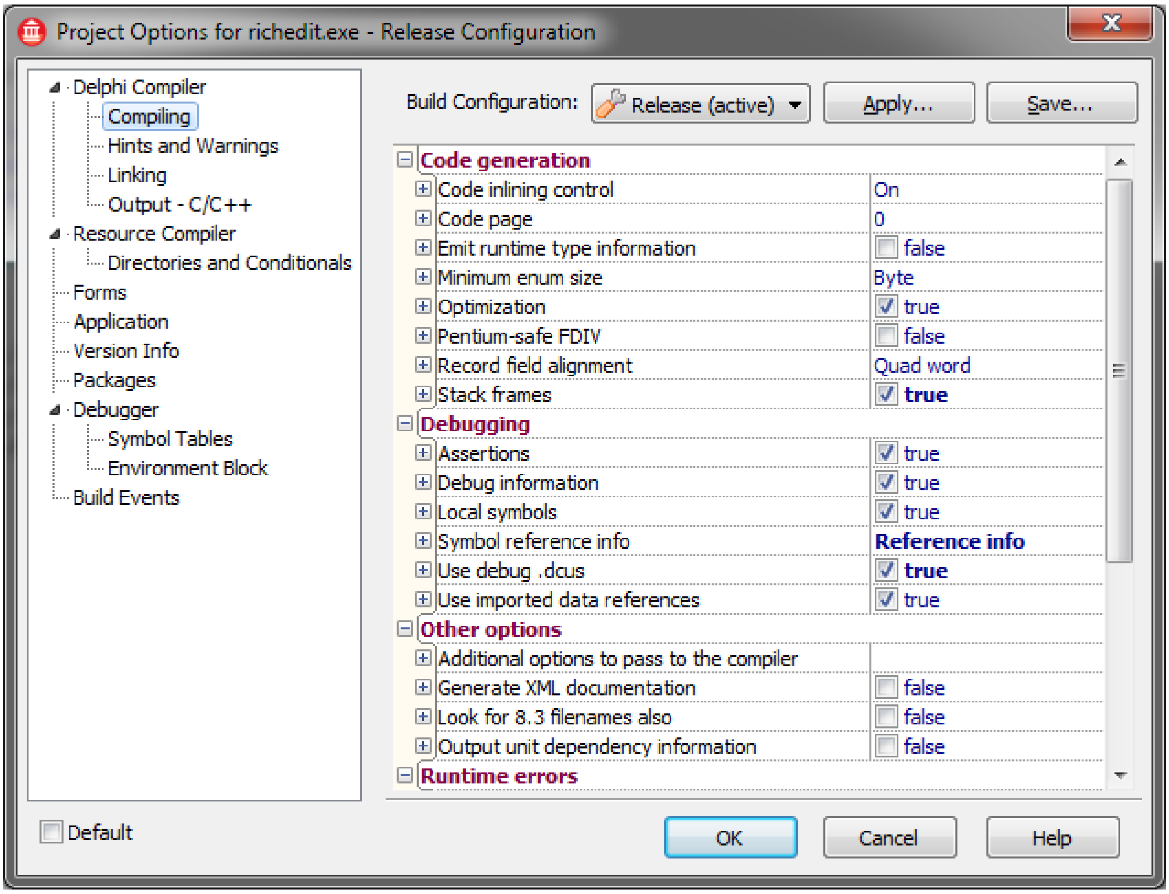
For best results it's recommended to disable packages in application (if possible) and turn off dynamic RTL - C++ only (if possible).

Long answer:

Note: we'll talk about **project options** in this article. We will *not* talk about **EurekaLog project options**. You can open project options via "Project" / "Options" IDE menu command.

Warning: please, remember that you need to make a **full build** (and not just "compile"), if you change any of these options.

We're interested in "Compiling" and "Linking" pages:

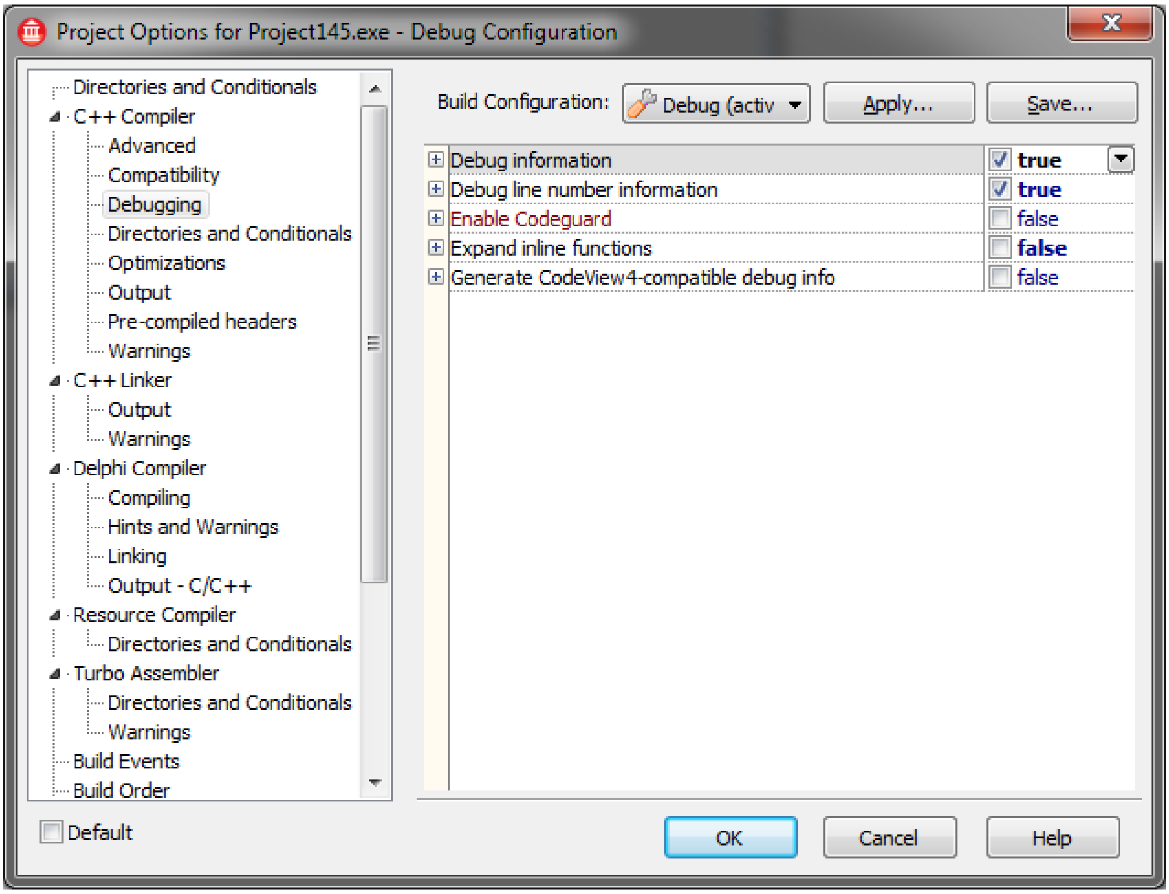


Build Configuration: Release (active)

Code generation	
Code inlining control	On
Code page	0
Emit runtime type information	□ false
Minimum enum size	Byte
Optimization	☑ true
Pentium-safe FDIV	□ false
Record field alignment	Quad word
Stack frames	☑ **true**
Debugging	
Assertions	☑ true
Debug information	☑ true
Local symbols	☑ true
Symbol reference info	**Reference info**
Use debug .dcus	☑ **true**
Use imported data references	☑ true
Other options	
Additional options to pass to the compiler	
Generate XML documentation	□ false
Look for 8.3 filenames also	□ false
Output unit dependency information	□ false
Runtime errors	

□ Default

Compiling page (Delphi)



Build Configuration: Debug (activ

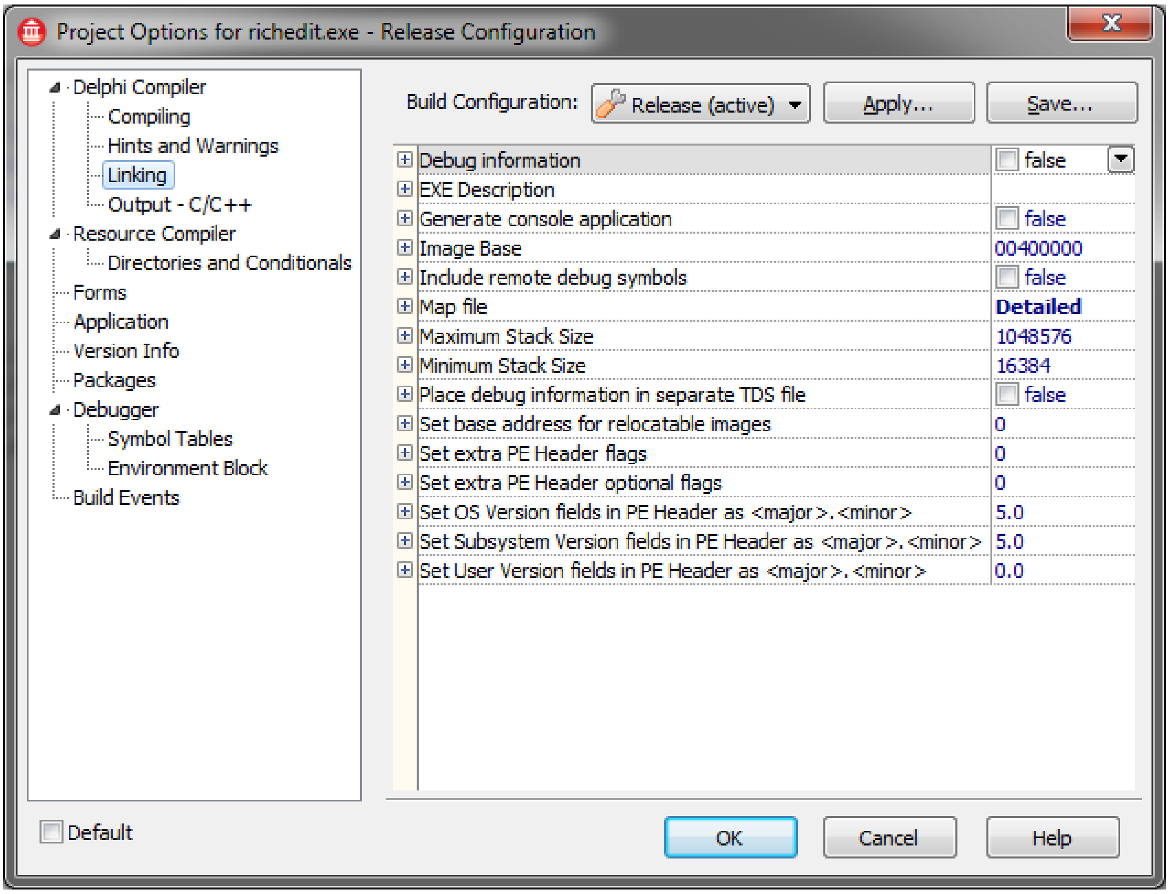
Debug information	☑ true
Debug line number information	☑ true
Enable Codeguard	□ false
Expand inline functions	□ false
Generate CodeView4-compatible debug info	□ false

□ Default

C++ Compiler/Debugging page (C++ Builder)

On "Compiling" page, we are interested in:

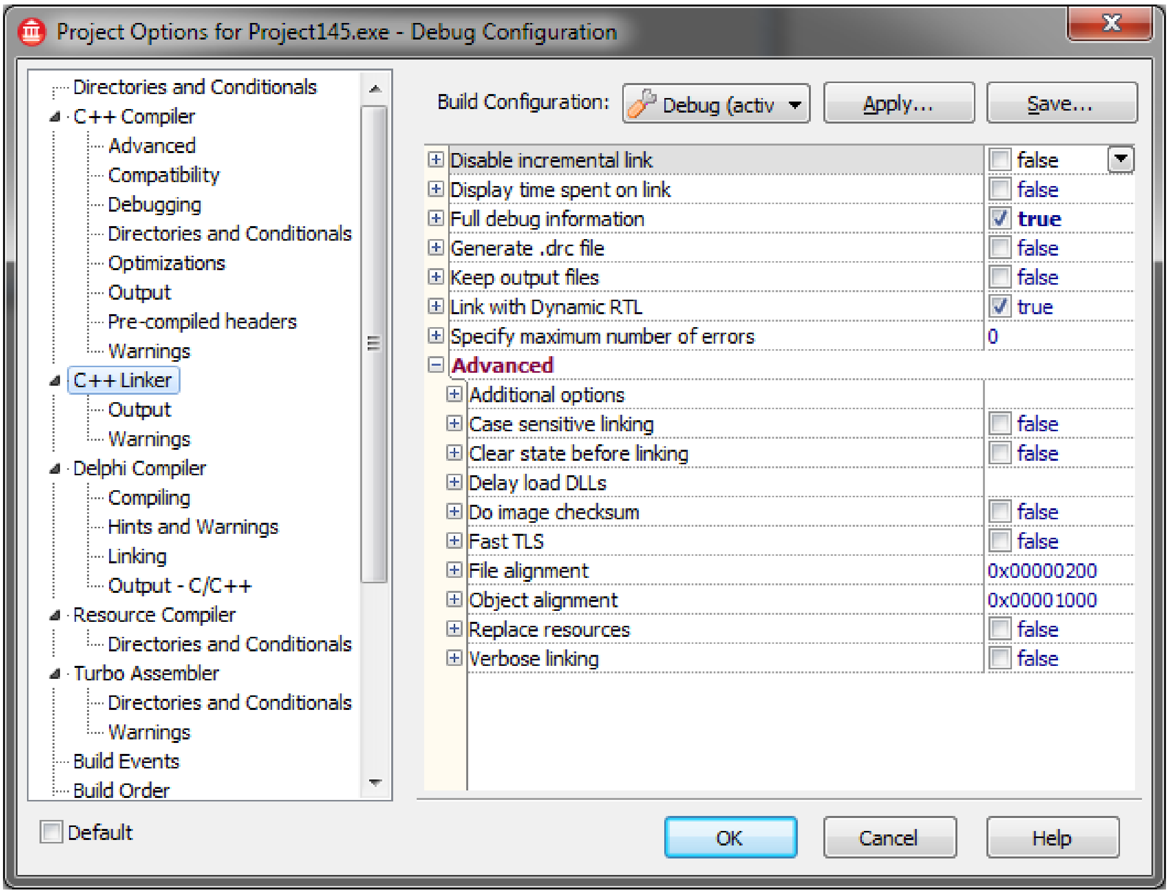
- Debug information + Local Symbols + Symbol reference info (Delphi only)
- Debug information + Debug line number info (C++ Builder only)
- Stack Frames (Delphi only)
- I/O Checking (Delphi only)
- Overflow checking (Delphi only)
- Range checking (Delphi only)



Build Configuration: Release (active)

Debug information	□ false
EXE Description	
Generate console application	□ false
Image Base	00400000
Include remote debug symbols	□ false
Map file	**Detailed**
Maximum Stack Size	1048576
Minimum Stack Size	16384
Place debug information in separate TDS file	□ false
Set base address for relocatable images	0
Set extra PE Header flags	0
Set extra PE Header optional flags	0
Set OS Version fields in PE Header as <major>.<minor>	5.0
Set Subsystem Version fields in PE Header as <major>.<minor>	5.0
Set User Version fields in PE Header as <major>.<minor>	0.0

Linking page (Delphi)

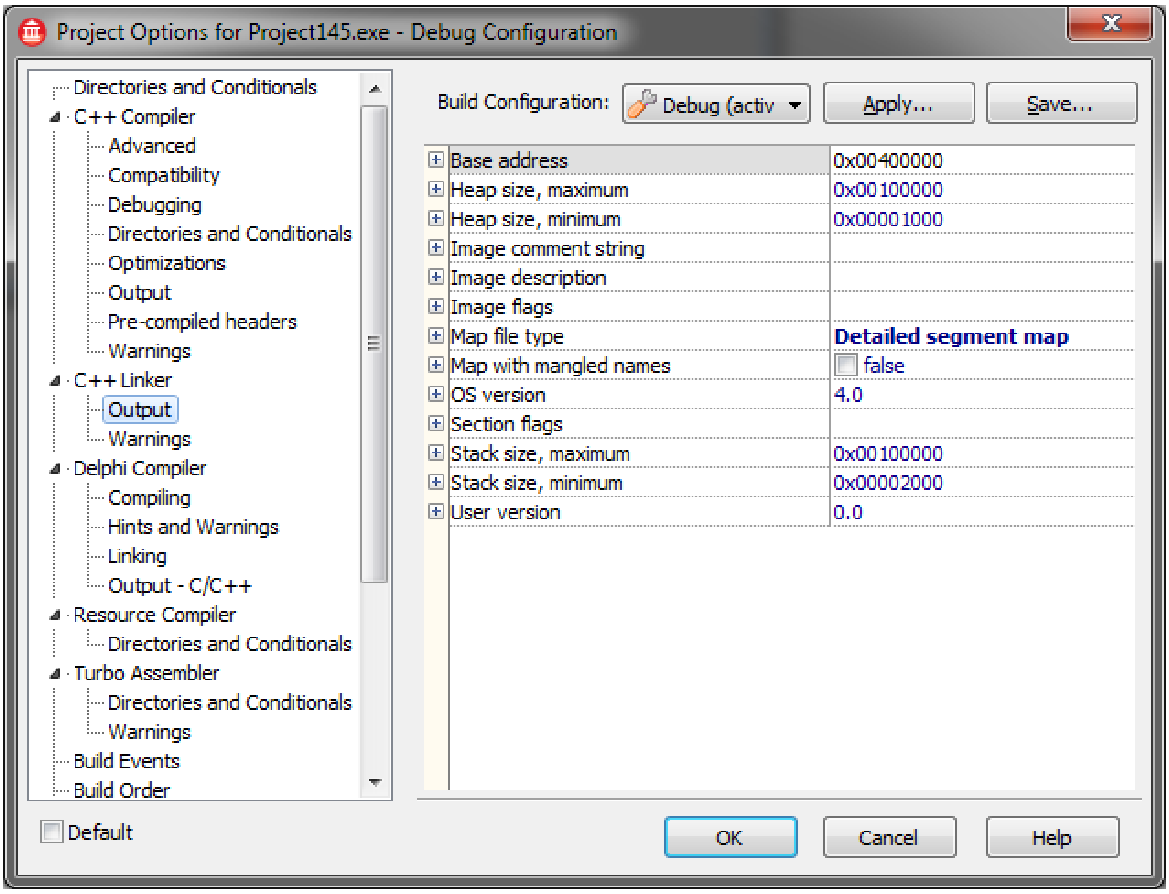


Build Configuration: Debug (activ

Disable incremental link	□ false
Display time spent on link	□ false
Full debug information	☑ true
Generate .drc file	□ false
Keep output files	□ false
Link with Dynamic RTL	☑ true
Specify maximum number of errors	0
Advanced	
Additional options	
Case sensitive linking	□ false
Clear state before linking	□ false
Delay load DLLs	
Do image checksum	□ false
Fast TLS	□ false
File alignment	0x00000200
Object alignment	0x00001000
Replace resources	□ false
Verbose linking	□ false

□ Default

C++ Linker page (C++ Builder)



⊞ Base address	0x00400000
⊞ Heap size, maximum	0x00100000
⊞ Heap size, minimum	0x00001000
⊞ Image comment string	
⊞ Image description	
⊞ Image flags	
⊞ Map file type	**Detailed segment map**
⊞ Map with mangled names	□ false
⊞ OS version	4.0
⊞ Section flags	
⊞ Stack size, maximum	0x00100000
⊞ Stack size, minimum	0x00002000
⊞ User version	0.0

C++ Linker/Output page (C++ Builder)

On "Linking" page, we are interested in:

- Map file
- (Full) Debug Information (this option is known as "Include TD32 debug info" in old IDE versions)
- Map with mangled names (C++ Builder only)
- Include remote debug symbols (Delphi only)

What do these options mean?

The most important settings are set of options "Debug information" ("Compiling" page).

The program is array of machine's (CPU) instructions - which are just bytes (numbers). The source code is a text file. The question: how does the debugger know, when he needs to stop, when you are setting a breakpoint in your source? Where is correspondence between raw numbers and human-readable text?

This correspondence is a debug information 40. Roughly speaking, the debug information is set of instructions like: "the machine codes no. 1056-1059 correspond to line 234 of Unit1". The debugger works thanks to such debug info.

And these options? They controls the generation of debug information for your units.

The debug information is stored in dcu-files together with its compiled code. I.e. the very same Unit1.pas can be compiled into different dcu-files (with or without debug information). The debug information increases compilation time, size of dcu-files, but it does not affect size or speed of resulting application (i.e. debug information is not included into application).

There are cases, when you want to have debug information in your files or (at least) near them. For example: if you are going to do remote debugging or debugging of external process. OR if you want to have human-readable call-stack in your exception diagnostic tool (EurekaLog).

EurekaLog takes care of injecting debug information into your executable. This work is performed by IDE expert 221 or command-line compiler 423.

However, project options affects detalization of debug information. So, you need to set project options properly to maximize result.

Let's do a short overview what these options do (see also: Compiling page and Linking page in Delphi's help and C++ Compiler page and C++ Linker page in C++ Builder's help). Most important options are marked in **bold**.

- "**Debug information**" – that is the debug info itself. You should enable this option if you want to do step-by-step debugging or have call stacks with names. EurekaLog IDE expert enables this option automatically for you. But don't forget to enable it, if you don't use IDE expert.
- "Local symbols" (Delphi only) – it is "addon" for usual debug information. This is correspondence between program's data and variables names in source code. You need to enable this option if you want to see and change variables. Also, call stack window in Delphi can display function's arguments with this option.
- "Reference info" (Delphi only) – this is additional information for code editor, which allows him to display detailed information about identificators. For example: where were a variable declared.
- "**Debug line number info**" (C++ Builder only) - it is "addon" for usual debug information. It adds information about line numbers in source files. EurekaLog IDE expert enables this option automatically for you. But don't forget to enable it, if you don't use IDE expert.

Those options are very close related and usually there is no need to enable or disable only one of them - they are switched together.

- "**Use Debug DCUs**" (Delphi only) - this very important option switches compilation between using debug and release versions of standard Delphi's units. If you were attentive, then you could notice that real pas-files in Delphi's Source folder are never used during compilation. Instead, the precompiled files (in dcu) are used. They are taken from Lib or Lib\Debug folders. This trick greatly decreases compilation time. Because dcu can be compiled with and without debug information - there are two sets of dcus in Lib folder. By toggling this option you'll specify which one Delphi should use for you. If you switch this option off - then you won't be able to debug standard Delphi code or see detailed call stack for it. EurekaLog doesn't enable this option for you, because it seriously alters debugging experience - you should enable this option manually for release production.
- "Stack Frames" (Delphi only) - this option controls stack frames generation. If the option is off then stack frames won't be generated unless they are needed. If the option is on - then stack frames will be generated always. Stack frames are used for frame-based stack-tracing method 578. I.e. it is used for building call stack. In usual application stack frames are generated almost everywhere.
- "Range checking" (Delphi only) - this is a very useful helper for debugging problems with array-based structures. With it, compiler will insert additional checks (for strings, arrays, etc), which checks the correctness of indexes. If you (by mistake) pass an invalid index - the exception of type ERangeError will be generated. And you can find your error. If the option is off then there is no additional code. Enabling this option slightly increases size of your application and slows down it execution. It is recommended to turn this option for debugging only. CodeGuard can serve as alternative to this option in C++ Builder.
- "Overflow checking" (Delphi only) - it is somehow similar to "Range checking", except checking code checks overflows in arithmetic operations. If result of operation is not suitable for storage variable - then exception EIntOverflow will be raised. For example: we have a byte variable, which holds 255 now. And we add 2 to it. There should be 257, but it can not be stored in byte variable, so real result will be 1. That is integer overflow.
- "I/O Checking" (Delphi only) - this option is used for working with "files in Pascal-style" (AssignFile, Reset, etc).
- "**Map file**" - by enabling this option you tell the Delphi's linker to create a separate .map file along with your executable. Map file contains human-readable representation of debug information. Different settings for this option controls the detalization level of output. Usually, there is no need to change it to anything, which differs from "Off" or

"Detailed". The map-file is used by various tools as primary source of debug information. For example, EurekaLog automatically turns this option on and uses map-file to create a debug information in its own format and then injects it into application. That is why you rarely need to change this option manually.

- "Debug Information" (Linker page, new Delphi)/"Include TD32 debug info" (old Delphi)/"**Full debug information**" (C++ Builder) - this option embeds debug information for external debugger in TD32 format into your application. You may need this option if you use "Run"/"Attach to process" and Delphi can not find debug information. Also, EurekaLog uses TD32 info to complete missing information in C++ Builder. Note, that size of your Delphi application can increase 5-10 times by enabling this option (C++ Builder writes information in separate .tds file), unless you enable "Place debug information in separate TDS file" option.
- "Include remote debug symbols" - very similar to previous option, but this creates a rsm-file with debug information for Delphi remote debugger. You need this option, if you want to do remote debugging.

Note, that some of these options can be enabled not only globally, but also separately for each unit (several options can affect single routines or, even, lines of code). This is done by using usual compiler directives (you can see them in help). For example, "Stack Frames" is controlled by {$W+} and {$W-} and Debug information from Compiling page is controlled by {$D+} and {$D-}.

So, we can give a recommendations for different use cases by summarizing all this info. These recommendations are written down below. Settings which differs from defaults are marked in **bold** (i.e. you should toggle items in bold manually).

Usual application, without EurekaLog

1. BASE SETTINGS FOR EACH PROFILE

- All debug options ("Debug information" (Compiler), "Local symbols", "Reference info") does not affect the resulting application and do not disturb us - so you usually should keep them always on.
- "Use Debug DCUs" - set it as you like (depending on: "do you want to debug standard Delphi code or not?").
- There is no need to turn on "Stack Frames" option - turn it off.
- There is no need for map-files - turn it off.

2a. DEBUG PROFILE

- **Turn ON** "Range checking" and (optionally) "Overflow checking".
- "Include TD32 debug info" - enable it only if you use "Attach to process" while debugging.
- "Include remote debug info" - enable it only if you want to use remote debugger.

2b. RELEASE PROFILE

- Turn off "Range checking", "Overflow checking", "Include TD32 debug info" and "Include remote debug info".

EurekaLog-enabled application

1. BASE SETTINGS FOR EACH PROFILE

- All debug options ("Debug information" (Compiler), "Local symbols", "Reference info") should be definitely turned on. Otherwise, call stack functionality won't be working.
- There is no need to turn off "Stack Frames" option - **turn it ON**.
- Generation of map-file should be **turned ON**, but EurekaLog's expert takes cares of it.
- Generation of TD32 information should be **turned ON** for C++ Builder, but EurekaLog's expert takes cares of it.

2a. DEBUG PROFILE

- "Use Debug DCUs" - set it as you like.
- **Turn ON** "Range checking" and (optionally) "Overflow checking".
- "Include TD32 debug info" - enable it only if you use "Attach to process" while debugging.
- "Include remote debug info" - enable it only if you want to use remote debugger.

2b. RELEASE PROFILE

- **Turn ON** "Use Debug DCUs" option.
- Turn off "Range checking", "Overflow checking", "Include TD32 debug info" and "Include remote debug info".

Note: if you don't do many index-based operations (so additional checks will not slow down your code) - then it may be a good idea to **always keep "Range checking" on**.

Note: your code or 3rd party code (i.e.: non-standard components) may use compiler directives in source code, which affect the options above. For example, a component may have {$D-} directive inside its .pas units to exclude itself from debugging. Obviously, doing so will prevent EurekaLog from getting descriptions for this component. You may need to change the source code and edit/remove such directives.

See also:

- Configuring bug report 46
- Configuring dialogs 52
- Configuring sending 53
- Customizing EurekaLog 180

Part VI

6 Typical scenarios

This section describes various scenarios of using EurekaLog. This should help you to get some ideas how to use it to help you troubleshot issues.

The described scenarios tells you about most common and typical uses. However, EurekaLog's capabilities are not limited to described scenarios. Sometimes our customers do some non-standard and even crazy things with EurekaLog. Check out our site (namely, blog and forums) to know some cool tricks.

Typical use of EurekaLog includes:

- Using EurekaLog to automate bug reporting (your's clients side) 68
- Using EurekaLog to debug bugs at developer's machine (your side) 70

6.1 Reporting

This article is part of Typical scenarios 68 series.

This is the primary target of EurekaLog. EurekaLog is exception tracer tool, which collects information about occurred problems (such as exceptions, hangs and leaks) in your application and notify you (as developer) about these issues.

Debugging in small scale

Debugging a single program run by a single user on a single computer is a well understood problem. It may be arduous, but follows general principles: when a user reproduces and reports an error, the programmer attaches a debugger to the running process and examines program state to deduce where algorithms or state deviated from desired behavior. When tracking particularly onerous bugs the programmer can resort to restarting and stepping through execution with the user’s data or providing the user with a version of the program instrumented to provide additional diagnostic information. Once the bug has been isolated, the programmer fixes the code and provides an updated program.

Debugging in large scale

Debugging in the large is harder. When the number of software components in a single system grows to the hundreds and the number of deployed systems grows to the millions, strategies that worked in the small, like asking programmers to triage individual error reports, fail. With hundreds of components, it becomes much harder to isolate the root cause of an error. With millions of systems, the sheer volume of error reports for even obscure bugs can become overwhelming. Worse still, prioritizing error reports from millions of users becomes arbitrary and ad hoc.

Back in old days programming teams struggled to scale with the volume and complexity of errors. Then there were tools invented, which could help to diagnose crashes in software, automatically collect a stack trace and upload this bug report to developer's server.

With EurekaLog data you can identify common real-world customer problems and provide a real-time solution to your customers. While customer support calls provide information about common issues, they do not always provide enough granular detail to debug the actual code. Further, support records indicate those problems which prompted calls — they do not indicate every instance of a crash.

Large number's law

Broad-based trend analysis of error reporting data shows that across all the issues that exist on the affected Windows platforms and the number of incidents received:

1. **80 percent of customer issues can be solved by fixing 20 percent of the top-reported bugs**.
2. **Addressing 1 percent of the top bugs would address 50 percent of the customer issues**.

The same analysis results are generally true on a company-by-company basis too (according to Microsoft's researches in error collecting).

If you could remove humans from the critical path and scale the error reporting mechanism to admit huge numbers of error reports, then you could use the law of large numbers to your advantage. For example, you didn't need to collect all error reports, just a statistically significant sample. And you didn't need to collect complete diagnostic samples for all occurrences of an error with the same root cause, just enough samples to diagnose the problem and suggest correlation. Moreover, once you had enough data to allow us to fix the most frequently occurring errors, then their occurrence would decrease, bringing the remaining errors to the forefront. Finally, even if you made some mistakes, such as incorrectly diagnosing two errors as having the same root cause, once you fixed the first then the occurrences of the second would reappear and dominate future samples.

Automate reporting

If you're waiting around for users to tell you about problems with your application, then you're seeing only a tiny fraction of all the problems that are actually occurring. Most users won't bother telling you about problems. They'll just quietly stop using your application.

That's why it's important to setup an exception and error reporting facility. It's your responsibility to ensure escape plan, if something will go wrong with your software. I.e. you not only need to protect users from errors, but you also need to protect yourself from your errors too. Errors are inevitable, and you must be prepared before they start happens. The situation will be pretty dire at this point, but some disaster recovery is possible, if you plan ahead.

You should also maintain a searchable and sortable database of errors somewhere. You need to have a central place where all of your errors are aggregated, a place which is visited by all your developers every day. Thus, bug reports will be de-facto TODO list for your team. You could also broadcast an error email notification to every developer. Or maybe have every crash automatically open a bug ticket in your bug tracking software.

Developing cycle

Once you have a detailed report on every crash, you can sort that data by frequency and spend your coding effort resolving the most common problems. Remember: fixing 20 percent of the top reported bugs solves 80 percent of customer issues.

If you don't have a central database of your bugs - then you can't sort bugs by "popularity". If you fix a bug that no actual user will ever encounter, what have you actually fixed? Given a limited pool of developer time, it's a way too better to allocate time toward fixing most hot problems.

1. Ship your software.
2. Get as many users as possible.
3. Study the bug reports, which users generate.
4. Use bug reports to focus on the most problematic areas of your code.
5. Fix few top bugs.
6. Make new version of your software, deploy it.
7. Repeat the process.

This data-driven feedback loop is so powerful you'll have (at least from the users' perspective) a rock-stable application in a sane number of iterations.

Conclusion

Automated bug reports are one of the most powerful form of feedback from your customers. The actual problems, with stack traces and other information, are collected for you, automatically and silently.

The sooner you can get your code out of your code editor and present it to real users - the sooner you'll have date to improve your software. Surely, it's very important to do as much as possible to fix bugs before shipping. The sooner you detect bug - the lower will be cost of its fixing.

However, your software will ship with bugs anyway. Everyone's software does. The question isn't how many bugs you will ship with, but how fast can you fix those bugs?

If your will practice the above mentioned approach (which is Exception Driven Development - EDD), the answer will be simple - you can improve your software almost in no time at all.

Note: the term "Exception Driven Development" was invented by Jeff Atwood.

See also:

- Managing bug reports in issue tracker 105
- Using EurekaLog for debugging 70

6.2 Debugging

This article is part of Typical scenarios 68 series.

Though EurekaLog's primary target is to collect reports "from the fields" 68, but many our customers use EurekaLog for other purposes. Most common side use is to debug bugs in applications directly on developer's machine. While Delphi and C++ Builder offer you a powerful debugger, which has many features and it's able to help you to solve almost any issue, but some people find EurekaLog to be more simple solution. I.e. they use EurekaLog to aid debugging on developer's machines.

Thus, EurekaLog offers some additional features, designed specifically to be used as part of developing and debugging stage. Thought they can be used in deployment release version of your software - it's highly unrecommended. If you do this - be sure to test your application to be comfortable for end-user.

Such "for debug only" features includes:

- Memory leaks checking 250 (please note that other memory features (non leaks related) are perfectly fine to be used in final release).
- Resource leaks checking 255.
- Handled exception catching 341.

Please note: since EurekaLog's primary target is to collect reports at run-time on your client's machine 68 - some of EurekaLog's features has lesser power than features in purely debugging software. For example, EurekaLog should use fast-enough approaches, as opposed to using heavy debugging code.

This means that sometimes it's better to use other software on your developer machine - such as Delphi's debugger (did you know that it has memory breakpoints?), debugging memory manager (FastMM in full debug mode, SafeMM, etc) or profiler (AQTime, etc). Those software is especially designed to be used as helpers at debugging stage. They typically are more powerful than EurekaLog, but because of this they can't be used as part of your release version - and that's EurekaLog's area 68.

Conclusion: typically you can try to use EurekaLog in debug version of your application to *quickly* locate issues. If EurekaLog is unable to help you - just switch to "heavy debugging" software.

Part VII

7 Solving bugs in your code

This set of articles is not related to EurekaLog, but helps you to find and solve bugs in your applications.

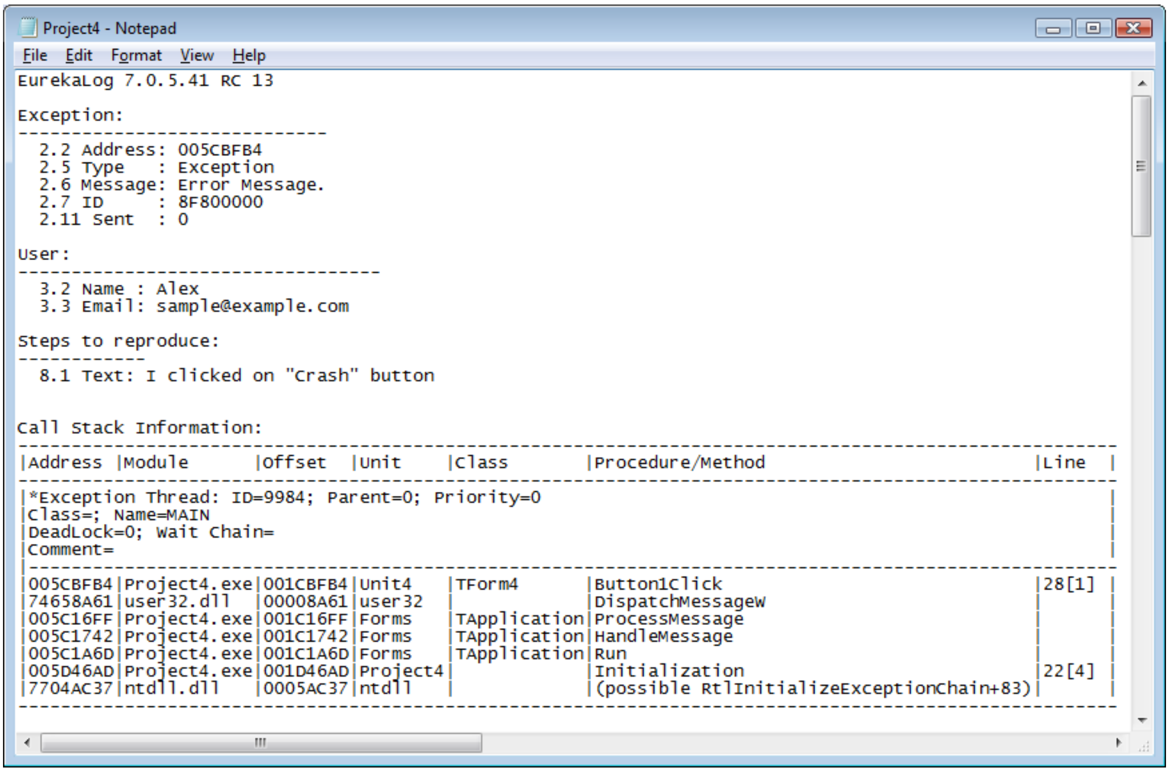
We have the following articles available:
- Understanding bug reports 72
  o Understanding call stacks 78
- Managing bug reports in issue tracker 105
- Common issues:
  - EAccessViolation 160
  - Leaks 166
  - Memory problems 171
  - Hangs and deadlocks 174

See also:
- Configuring call stack 48

7.1 Bug reports

Bug report is a report about possible **Bug** in the application. It usually contains information about **Exception** or **Leak**. **Bug report** usually contains one or more **Call stacks** and some information about system itself and application.

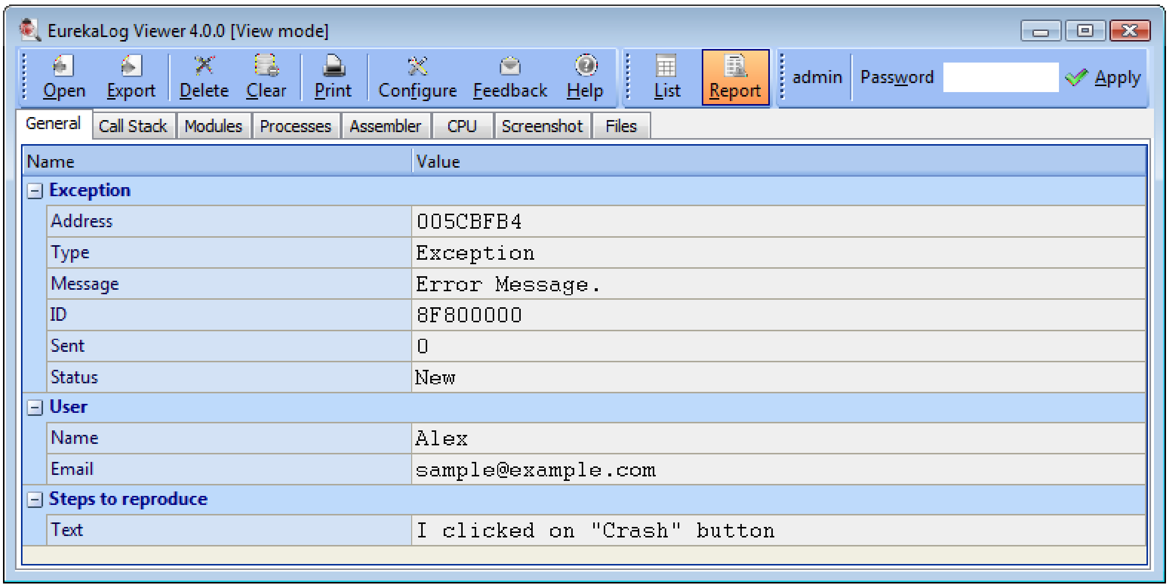
Bug reports are stored in files. EurekaLog's **Bug reports** have .el, .elp, .elx file extensions. The .el files are usual text files, which can be opened in any text editor:



```
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 : Alex
  3.3 Email: sample@example.com

Steps to reproduce:
------------
  8.1 Text: I clicked on "Crash" button


Call Stack Information:
-------------------------------------------------------------------------------------------------------
|Address |Module      |Offset  |Unit    |Class       |Procedure/Method                         |Line  |
-------------------------------------------------------------------------------------------------------
|*Exception Thread: ID=9984; Parent=0; Priority=0                                                     |
|Class=; Name=MAIN                                                                                    |
|DeadLock=0; Wait Chain=                                                                              |
|Comment=                                                                                             |
|-----------------------------------------------------------------------------------------------------
|005CBFB4|Project4.exe|001CBFB4|Unit4   |TForm4      |Button1Click                             |28[1] |
|74658A61|user32.dll  |00008A61|user32  |            |DispatchMessageW                         |      |
|005C16FF|Project4.exe|001C16FF|Forms   |TApplication|ProcessMessage                           |      |
|005C1742|Project4.exe|001C1742|Forms   |TApplication|HandleMessage                            |      |
|005C1A6D|Project4.exe|001C1A6D|Forms   |TApplication|Run                                      |      |
|005D46AD|Project4.exe|001D46AD|Project4|            |Initialization                           |22[4] |
|7704AC37|ntdll.dll   |0005AC37|ntdll   |            |(possible RtlInitializeExceptionChain+83)|      |
-------------------------------------------------------------------------------------------------------
```

Notepad showing content of Project4.el file

Textual bug reports (.el) contain human-readable description of the problem, which can be analyzed by developer to get the idea what was going wrong. Such reports can be saved locally or sent to developers. Such reports can be viewed in any text viewer/editor. However, you can also use EurekaLog Viewer tool 617 to view these files in a more convenient way:



Name	Value
Exception	
Address	005CBFB4
Type	Exception
Message	Error Message.
ID	8F800000
Sent	0
Status	New
User	
Name	Alex
Email	sample@example.com
Steps to reproduce	
Text	I clicked on "Crash" button

EurekaLog Viewer tool showing content of the same Project4.el file

Textual bug reports can hold only text. You can't add files to such bug reports. .elp (EurekaLog Packed) format is capable of holding both text report and any additional files - such as screenshots, logs, currently opened files, etc. Packed bug reports (.elp) can not be viewed by text editors - because this bug report format is binary format. These bug reports can be viewed in EurekaLog Viewer tool. .elp files can be sent to developers, these files are not saved locally.

Notes:

- .elp-file is renamed zip-file which holds .el report and any additional files;
- .elp-file is used only for sending bug reports to developers (via e-mail, bug tracker, etc.). Local bug reports can be only in .el format. This means that you can not include screenshot or any additional files into local report.

Bug report contain information about running application when it encountered a bug. This information can be split in the following categories:

- Information about bug (exception or leak) itself - such as error message, location's address, CPU state, etc. This information is most important in bug reports and it's used to resolve bug.
- Information about application in general - such as version information of executable, name, description, etc. This information is used to identify application which encountered a bug.
- Information about current user - such as name, e-mail, etc. This information is used to identify user that sent the report. This information can also be used to contact user later for more infos.
- Information about run-time environment - such as loaded DLLs, installed hardware, running processes, etc. This is auxiliary information which is used to make guesses when searching for bug's reason.

All of the above can also be divided into two mutually exclusive groups:

- Immediate. This information is captured when the problem occurs. This includes error message and type, address, call stack, CPU state, etc.
- Delayed. This information is captured when bug report is created. Examples are list of loaded DLLs, running modules, hardware info, etc.

Immediate information is collected right away (when exception raised or leak found) - because it's very important to capture as much of detailed and precise information about the problem as possible. On the other hand, capturing any piece of information takes time. Thus it's not wise to capture ALL information right when exception or leak is detected - due to performance reasons. Not all exceptions will be reported as bugs, and leaks are usually grouped into a single report. That's why some information is captured later: when final bug report is created. This information doesn't change very rapidly (harware info, running processes, etc.), so a bit later snapshot will be close enough.

EurekaLog's bug report contain the following sections:

- General 74 - this section includes all non-list (i.e. single valued) information: error message, user's infor, computer's info, etc. All other sections contain specialized lists;
- Call Stack 78 - this section includes one or more call stacks. Call stack is a sequence of addresses (mostly with textual description) which leads us to place of the problem;
- Modules 99 - this section includes list of the loaded DLLs in the current process;
- Processes 99 - this section includes list of running processes on the current machine;
- Assembler 100 - this section includes machine code listing near problem's location;
- CPU 103 - this section includes CPU dump;
- Screenshot 105 (.elp only) - this section shows screenshot of a running application when it encountered a bug;
- Files 105 (.elp only) - this section shows list of attached files.

Usually, analyzing of bug report is starting with reading and understanding call stack 78. However, there are some specific issues when you need to start with auxiliary information in bug report - such as leaks 166.

See also:

- Configuring bug report 46
- Configuring call stack 48
- Managing bug reports in issue tracker 105
- How to save/capture ZIP/ELP file
- How to always create ZIP/ELP file

7.1.1 General section

Any EurekaLog's bug report contain the general section which holds single-valued information:

```
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 : Alex
  3.3 Email: sample@example.com

Steps to reproduce:
------------
  8.1 Text: I clicked on "Crash" button
```

Example of a short General section

Exactly which information will be included into bug report is configured in options 266. Exception's address, type, message and BugID (2.2, 2.5, 2.6, and 2.7) are mandatory pieces in any EurekaLog's bug report and can not be removed. Any other information can be enabled or disabled:

```
Application:
-------------------------------------------------------
  1.1 Start Date      : Sat, 17 Aug 2013 19:15:47 +0400
  1.2 Name/Description: Project10.exe
  1.3 Version Number  : 1.0.1.15
  1.4 Parameters      : /TestCrash
  1.5 Compilation Date: Sat, 17 Aug 2013 19:15:30 +0400
  1.6 Up Time         : 2 second(s)

Exception:
-----------------------------------------------------
  2.1 Date          : Sat, 17 Aug 2013 19:15:50 +0400
  2.2 Address       : 00000000007F0D43
  2.3 Module Name   : Project10.exe
  2.4 Module Version: 1.0.1.15
  2.5 Type          : ETestException
  2.6 Message       : This is a test exception.
  2.7 ID            : 68260000
  2.8 Count         : 1
  2.9 Status        : New
  2.10 Note         :
  2.11 Sent         : 0

User:
-------------------------------------------------------
  3.1 ID        : Alex
  3.2 Name      : Alexander
  3.3 Email     : sample@example.com
  3.4 Company   : Neos Eureka S.r.l.
  3.5 Privileges: SeShutdownPrivilege             - OFF
                  SeChangeNotifyPrivilege         - ON
                  SeUndockPrivilege               - OFF
                  SeIncreaseWorkingSetPrivilege   - OFF
                  SeTimeZonePrivilege             - OFF

Active Controls:
----------------------------
  4.1 Form Class   : TForm9
  4.2 Form Text    : Form9
  4.3 Control Class: TButton
  4.4 Control Text : Button7

Computer:
-------------------------------------------------------------------------------
-----
  5.1 Name            : TEST-PC
  5.2 Total Memory    : 4293558272 (4.00 Gb)
  5.3 Free Memory     : 1398812672 (1.30 Gb)
  5.4 Total Disk      : 274875805696 (256.00 Gb)
  5.5 Free Disk       : 128265609216 (119.46 Gb)
  5.6 System Up Time  : 6 hour(s), 38 minute(s), 25 second(s)
  5.7 Processor       : Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz
  5.8 Display Mode    : 1827 x 1000, 32 bit
  5.9 Display DPI     : 96
  5.10 Video Card      : VirtualBox Graphics Adapter (driver 4.2.14.0 - RAM
67108864)
  5.11 Printer        : Microsoft XPS Document Writer (driver 6.0.6002.18005)
  5.12 Virtual Machine: VirtualBox

Operating System:
------------------------------------------------
  6.1 Type    : Microsoft Windows Vista (64 bit)
  6.2 Build # : 6002
  6.3 Update  : Service pack 2
  6.4 Language: English
  6.5 Charset : 0

Network:
```

```
--------------------------------
  7.1 IP Address: 192.168.001.243
  7.2 Submask   : 255.255.255.000
  7.3 Gateway   : 192.168.001.001
  7.4 DNS 1     : 192.168.001.001
  7.5 DNS 2     : 000.000.000.000
  7.6 DHCP      : ON

Steps to reproduce:
------------
  8.1 Text: I clicked on "Crash" button
```

Example of a large General section

Usually, information in general section is used to identify application, crashed module, error (type/message), and user. It also contains bunch of run-time information (such as hardware info, OS info, etc.). Run-time information (along with modules 99 and processes 99) serves as auxiliary information when analyzing call stack 78 and assembler 100/CPU dump 103.

All captions can be localized in options 339. Values are identified by numbers (like 1.1, 1.2, ...8.1). Values are non-localizable. All values are gathered through OS's information functions and appears "as is" in bug reports.

All date-time values are shown in local time zone. There is time-zone shift attached to each date-time value ("+0400" in the example above - which reads as "+4 hours 00 minutes from GMT/UTC, i.e. subtract 4 hours to get UTC time").

Memory and disk values are shown in precise and compact form. Precise form is used for machine or when you need exact value. Compact form is used as more human-readable - to get estimate quickly.

Application block displays information about application - that is .exe file. If your project is a DLL - then this block will show information about host process, not about your DLL.

- Start Date value shows when application was launched. This values is saved on application's startup and is inserted into bug reports unmodified;
- Name/Description value shows .exe's file name (without path) and description extracted from version information (if available). "File description" field is used as source for description. Full path can be obtained from Modules section 99 of bug report;
- Version Number value shows .exe's version information (if available). dwFileVersion field from VS_FIXEDFILEINFO structure is used as source;
- Parameters value shows any command-line arguments passed to application (without exe file itself). This value comes from GetCommandLine function;
- Compilation Date value shows date/time when application was compiled. This information is stored inside executable when it's compiled;
- Up Time value shows how long application is running. This is delta between Start Date value and current time. This value is not stored anywhere, it is calculated when bug report is created.

Note: EurekaLog duplicates "Compilation Date" value inside its information block injected inside executable. Thus, you will get exact and correct compilation time value even if exe header was modified. Please note that some compilers (including old Borland compilers) fails to setup a proper compilation time in PE headers. This means that you may get strange values from non-EurekaLog enabled files (for example, when running your DLL inside non-EurekaLog host).

Exception block contains information about raised exception. This block will contain dummy values for hangs and leaks.

- Date value shows exact moment when exception (or leak, or hang) was detected;
- Address value shows address of machine instruction which raised the exception. Note that this is logical exception address, see CPU section 103 for description;
- Module Name shows file name and description for module to which Address value belongs. This value is formed by the same rules as Name/Description value from Application block (see above), but it's gathered from a module which raised exception - which may be or may be not a main exe file;

- Module Version value is the same as Version Number value from Application block, but it is taken from exception module;
- Type value is class name of exception object;
- Message value is error message - that is a value of Message property from exception object;
- ID value shows Bug ID 421. Bug ID is a hash which unique identifies exception/leak/hang;
- Count value shows how many times this exception was raised. Typically this value is 1, but it can be more if this exception was raised multiple times before bug report was sent. Please note that this value is not affected by other exceptions raised before this one (that is exceptions with other Bug IDs);
- Status value indicate bug tracker's status of the exception. This is obsolete value which exists for backward-compatibility reasons. Older versions of EurekaLog used plain text files as bug database, and this field was used as "New/Opened/Resolved" column;
- Note value is the same as Status value - it's obsolete. It was used to hold arbitrary remarks about exception in plain-text database;
- Sent value indicate if this report was sent to developers or not. 0 indicate that report was not sent.

User block contains information about user account running application:

- ID value shows current user account name. This value comes from GetUserName function;
- Name value shows full user name (if possible). This value is gathered from multiple sources in the registry;
- Email value shows e-mail of the user. This value is supplied by MS Classic 377 error dialog;
- Company value shows "Registered Company" value gathered from the registry;
- Privileges value shows account privileges of running process;

Active Controls block shows information about keyboard focus in GUI applications:

- Form Class value shows window class of foreground window. Foreground window is determinated by calling GetForegroundWindow function;
- Form Text value shows caption of foreground window;
- Control Class value shows window class of focused control. Focused control is determinated by calling GetFocus function;
- Control Text value shows caption/text of focused control.

Computer block shows hardware information. Values in this block are self-explanatory.

Note: Virtual Machine value is filled only when application is running outside of the debugger. That's because probing for virtual machines requires application to try to execute invalid instructions which will raise exceptions (such as "Invalid Instruction" or "Privileged instruction") when application is run on real hardware. So, virtual machine checks are disabled when application is debugged to reduce annoying from raised probing exceptions.

Operating System block shows OS information:

- Type value shows human-readable name of OS. This value can only show OS known to EurekaLog when application was compiled. This field will show a generic "Windows 8.7" (this is just an example) for any unknown OS version;
- Build value shows OS build. This value comes from dwBuildNumber field of GetVersionEx function;
- Update value shows installed service pack. This value comes from wServicePack field of GetVersionEx function;
- Non-Unicode Language value shows string representation of GetSystemDefaultLangID function. This value is known as "Language for non-Unicode programs" for the user;
- Charset/ACP value shows default user locale charset and ANSI code page. These are value of ciCharset and ciACP fields of CharsetInfo structure which is obtained through GetLocaleInfo function for default code page (LOCALE_IDEFAULTANSICODEPAGE) of LOCALE_USER_DEFAULT locale. ciACP field matches GetACP function.
- Install language value shows string representation of GetSystemDefaultUILanguage function. This value is known as "Install Language" for the system;
- UI language value shows string representation of GetUserDefaultUILanguage function. This value is known as "UI Language" for the user;

Note: Microsoft can return false information for backward compatibility reasons. For example, new versions of OS may report themselves as sub-versions of previous OS version. Or it may report service pack 2 being installed when there was no SP released at all (SP 2 is a most popular SP for Windows XP). Application Compatibility layer of OS may also affect these values.

Network block shows network information. Values in this block are self-explanatory. Each network adapter will produce a vertical section of values.

Steps to reproduce block shows feedback text from user:
- Text value is supplied by MS Classic 377 or EurekaLog 379 error dialogs. Usually this is "steps to reproduce" text - that is a textual description entered by end user which explains what user was doing when error occurred.

Custom information block shows information supplied by your code via OnCustomDataRequest event handler. Please note that you still have to enable custom information in bug report content options 266.

See also:
- Understanding bug reports 72
- Understanding call stacks 78

7.1.2 Call Stack section

Call stack - stack data structure that stores information about the active subroutines of the application. This kind of stack is also known as an *execution stack*, *control stack*, *run-time stack*, or *machine stack*, and is often shortened to just "*the stack*". **Call stack** is a central piece of information in any **Bug report**. **Call stack** is a sequence of addresses (mostly with textual description), which leads us to place of the problem (**Exception** or **Leak**).

EurekaLog generates bug reports 72 which can be saved to file 46 and/or send to developers 53 (you). Default form of bug report is a text file with .el extension. It can be opened in any text editor. EurekaLog also has Viewer tool which helps you to work with bug reports. Bug report can be also stored in packed form (.elp file) or XML-encoded form (.elx).

No matter which form bug report takes - it always contains the same information. The central part of any bug report is a call stack 79.

```
Call Stack Information:
---------------------------------------------------------------------
|Address |Module      |Unit    |Class       |Procedure/Method|Line  |
---------------------------------------------------------------------
|*Exception Thread: ID=9984; Parent=0; Priority=0                   |
|Class=; Name=MAIN                                                  |
|DeadLock=0; Wait Chain=                                            |
|Comment=                                                           |
|--------------------------------------------------------------------
|005CBFB4|Project4.exe|Unit4   |TForm4      |Button1Click    |28[1] |
|74658A61|user32.dll  |user32  |            |DispatchMessageW|      |
|005C16FF|Project4.exe|Forms   |TApplication|ProcessMessage  |      |
|005C1742|Project4.exe|Forms   |TApplication|HandleMessage   |      |
|005C1A6D|Project4.exe|Forms   |TApplication|Run             |      |
|005D46AD|Project4.exe|Project4|            |Initialization  |22[4] |
---------------------------------------------------------------------
```

Example of a typical call stack in bug report

There may be multiple call stacks in a single bug report. There is always one call stack minimum - for exception's thread, for hanged thread, or for a leaked resource. But there can be more than one. For example, multi-threaded application can include call stacks of other running threads. And there may be more than one leak, so there will be one call stack per each leak in bug report.

Call stacks can be configured in corresponding options dialog 237.

Working with any bug report usually goes in this way:
1. You start with understanding call stack 81
2. You search for bug's location 95
3. You search for bug's reason 97

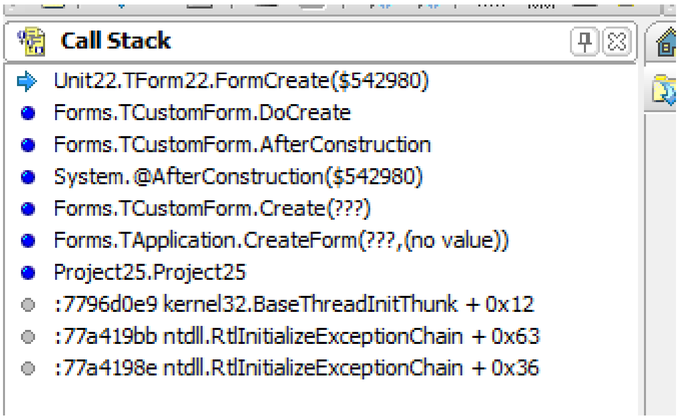
See also:
- Configuring call stack 48
- Understanding bug reports 72

7.1.2.1 Call stacks

This article is a part of working with bug reports 72.

Call stack 40 is a central piece of information in any bug report. Call stack is a sequence of addresses (mostly with textual description), which leads us to place of the problem (exception or memory problem).

Here are examples of call stacks in different forms:

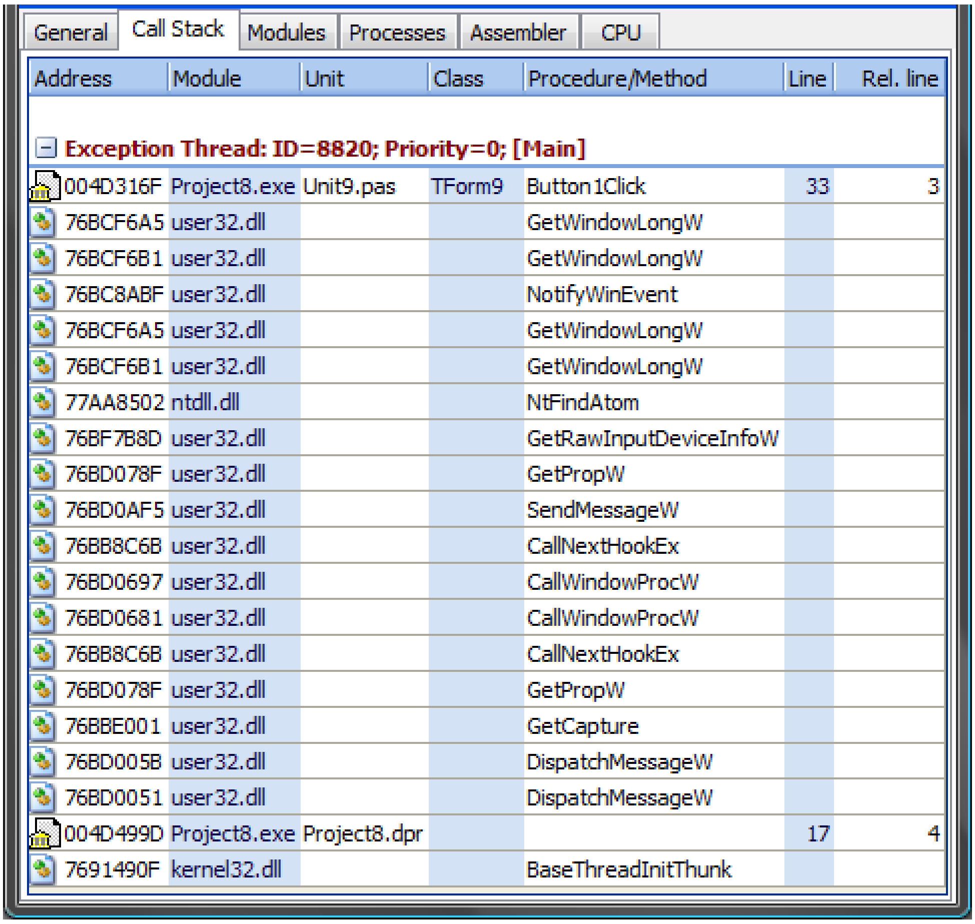


```
Unit22.TForm22.FormCreate($542980)
Forms.TCustomForm.DoCreate
Forms.TCustomForm.AfterConstruction
System.@AfterConstruction($542980)
Forms.TCustomForm.Create(???)
Forms.TApplication.CreateForm(???,(no value))
Project25.Project25
:7796d0e9 kernel32.BaseThreadInitThunk + 0x12
:77a419bb ntdll.RtlInitializeExceptionChain + 0x63
:77a4198e ntdll.RtlInitializeExceptionChain + 0x36
```

Delphi's call stack (*View/Debug Windows/Call Stack*)

```
Call Stack Information:
--------------------------------------------------------------------------------
|Address |Module      |Unit         |Class |Procedure/Method       |Line |
--------------------------------------------------------------------------------
|*Exception Thread: ID=8820; Priority=0; Class=; [Main]                         |
|-------------------------------------------------------------------------------|
|004D316F|Project8.exe|Unit9.pas    |TForm9|Button1Click           |33[3]|
|76BCF6A5|user32.dll  |             |      |GetWindowLongW         |     |
|76BCF6B1|user32.dll  |             |      |GetWindowLongW         |     |
|76BC8ABF|user32.dll  |             |      |NotifyWinEvent         |     |
|76BCF6A5|user32.dll  |             |      |GetWindowLongW         |     |
|76BCF6B1|user32.dll  |             |      |GetWindowLongW         |     |
|77AA8502|ntdll.dll   |             |      |NtFindAtom             |     |
|76BF7B8D|user32.dll  |             |      |GetRawInputDeviceInfoW|     |
|76BD078F|user32.dll  |             |      |GetPropW               |     |
|76BD0AF5|user32.dll  |             |      |SendMessageW           |     |
|76BB8C6B|user32.dll  |             |      |CallNextHookEx         |     |
|76BD0697|user32.dll  |             |      |CallWindowProcW        |     |
|76BD0681|user32.dll  |             |      |CallWindowProcW        |     |
|76BB8C6B|user32.dll  |             |      |CallNextHookEx         |     |
|76BD078F|user32.dll  |             |      |GetPropW               |     |
|76BBE001|user32.dll  |             |      |GetCapture             |     |
|76BD005B|user32.dll  |             |      |DispatchMessageW       |     |
|76BD0051|user32.dll  |             |      |DispatchMessageW       |     |
|004D499D|Project8.exe|Project8.dpr|       |                     |17[4]|
-----------------------------------------------------------------------
```

EurekaLog’s call stack (raw text)



⊟ Exception Thread: ID=8820; Priority=0; [Main]

Address	Module	Unit	Class	Procedure/Method	Line	Rel. line
004D316F	Project8.exe	Unit9.pas	TForm9	Button1Click	33	3
76BCF6A5	user32.dll			GetWindowLongW		
76BCF6B1	user32.dll			GetWindowLongW		
76BC8ABF	user32.dll			NotifyWinEvent		
76BCF6A5	user32.dll			GetWindowLongW		
76BCF6B1	user32.dll			GetWindowLongW		
77AA8502	ntdll.dll			NtFindAtom		
76BF7B8D	user32.dll			GetRawInputDeviceInfoW		
76BD078F	user32.dll			GetPropW		
76BD0AF5	user32.dll			SendMessageW		
76BB8C6B	user32.dll			CallNextHookEx		
76BD0697	user32.dll			CallWindowProcW		
76BD0681	user32.dll			CallWindowProcW		
76BB8C6B	user32.dll			CallNextHookEx		
76BD078F	user32.dll			GetPropW		
76BBE001	user32.dll			GetCapture		
76BD005B	user32.dll			DispatchMessageW		
76BD0051	user32.dll			DispatchMessageW		
004D499D	Project8.exe	Project8.dpr			17	4
7691490F	kernel32.dll			BaseThreadInitThunk		

EurekaLog’s call stack (**EurekaLog Viewer**)

```
4CC5B7 [Unit15.pas][Unit15][Unit15.B][35]
4CC5C4 [Unit15.pas][Unit15][Unit15.A][39]
4CC5DC [Unit15.pas][Unit15][Unit15.TForm15.FormCreate][43]
4C0CCB [Forms][Forms.TCustomForm.DoCreate]
4C0913 [Forms][Forms.TCustomForm.AfterConstruction]
4C08E8 [Forms][Forms.TCustomForm.Create]
4CA539 [Forms][Forms.TApplication.CreateForm]
4CD986 [Project14][Project14.Project14][14]
75B94911 [BaseThreadInitThunk]
770FE4B6
```

FastMM’s call stack

```
(00087A8C){Project1.exe} [00488A8C] Unit1.B (Line 40, "Unit1.pas" + 1) + $5
(00087ABC){Project1.exe} [00488ABC] Unit1.A (Line 44, "Unit1.pas" + 0) + $0
(00055523){Project1.exe} [00456523] Controls.TWinControl.WndProc + $513
(0003AA8C){Project1.exe} [0043BA8C] StdCtrls.TButtonControl.WndProc + $6C
(00055673){Project1.exe} [00456673] Controls.DoControlMsg + $23
(00055523){Project1.exe} [00456523] Controls.TWinControl.WndProc + $513
(00068584){Project1.exe} [00469584] Forms.TCustomForm.WndProc + $594
(00054C3C){Project1.exe} [00455C3C] Controls.TWinControl.MainWndProc + $2C
(00025724){Project1.exe} [00426724] Classes.StdWndProc + $14
```

```
(0005561F){Project1.exe} [0045661F] Controls.TWinControl.DefaultHandler + $D7
(00055523){Project1.exe} [00456523] Controls.TWinControl.WndProc + $513
(0003AA8C){Project1.exe} [0043BA8C] StdCtrls.TButtonControl.WndProc + $6C
(00025724){Project1.exe} [00426724] Classes.StdWndProc + $14
```

JCL's call stack

```
main thread ($7c):
3ecf8ea9 +0000 mshtml.dll
7c90e470 +0010 ntdll.dll KiUserCallbackDispatcher
7e42b1a6 +000a USER32.dll DestroyWindow
0086ad33 +001f Project1.exe OleCtrls 640 +11 TOleControl.Destroy
008715c9 +00a1 Project1.exe DHTMLEdit 5225 +16 TCustomDHTMLEdit.Destroy
00533c11 +008d Project1.exe Controls 7737 +16 TWinControl.Destroy
004ee915 +0039 Project1.exe ComCtrls 5666 +6 TTabSheet.Destroy
00533c11 +008d Project1.exe Controls 7737 +16 TWinControl.Destroy
004edb52 +005e Project1.exe ComCtrls 5144 +5 TCustomTabControl.Destroy
004eef15 +0049 Project1.exe ComCtrls 5889 +3 TPageControl.Destroy
00533c11 +008d Project1.exe Controls 7737 +16 TWinControl.Destroy
00510770 +0028 Project1.exe Forms 2644 +3 TScrollingWinControl.Destroy
005116a9 +00f9 Project1.exe Forms 3246 +33 TCustomForm.Destroy
7c90e485 +0009 ntdll.dll KiUserExceptionDispatcher
7ca2b137 +00b4 SHELL32.dll SHGetFileInfoW
```

madExcept's call stack

So, as you can see: call stack always contains very similar information, which don't depend on tool created it (well, there are all different applications in the above examples, so don't try to compare them).

Now it's time to learn how to read call stacks 81.

See also:

- Reading and understanding bug reports 72
- Configuring call stack 48
- How to read call stack 81
- Multi-threaded call stacks 85
- Call stack formats 83
- Searching bug's location 95
- Final notes on call stack 96
- Searching bug's reason 97

7.1.2.2 How to read call stacks

This article is a part of working with bug reports 72.

Call stack is set of lines, which is usually read from top to bottom - meaning moving from current locations to callers. The bottom line was executed first. The top line is executed last and it is the current routine.

For example, if you have this call stack (short pseudo form):

- DoSomething
- DoWork
- Run

This means that `Run` procedure was executed first. `Run` called `DoWork`, which in turn called `DoSomething`. `DoSomething` encountered an error - and this call stack was created. The code for such call stack may look like this:

```
procedure DoSomething;
begin
  PInteger(nil)^ := 0; // <- exception here,
  // call stack was created from this point
end;

procedure Run;

  procedure DoWork;
  begin
    DoSomething;
  end;

begin
  DoWork;
end;
```

In other words, this call stack means that execution of your code goes through this way: Run -> DoWork -> DoSomething -> error.

Each line in call stack can contain this information (some parts are optional and may be missed in the specific call stack format):

- **Code address**. Those are values like 004D316F or 76BCF6A5 (and also with '$' or '0x' prefix - like $004D316F or 0x004D316F). This is an absolute address in HEX-form of the code, which contains a routine's call. This value is present almost always, unless it's replaced with textual description (unit and routines names).
- **Stack address**. Those are values like 0018F628 or 0228FF84. It is an address (in HEX format) inside thread's stack, where code address was found. This value is present very rarely. Usually it follows or precedes code address.
- **Executable module name**. Those are values like 'Project1.exe' and 'ntdll.dll'. It's a file name of executable module, which contains code address. In rare cases can be '' (empty string) - that is, the code doesn't belong to any module (like run-time generated, self-modified code, etc). Such case is rare in typical application.
- **Code offset**. Those are values like 000D316F or +D316F. It's a difference (usually in HEX format) between code address and base address of executable module name. For example, base address of exe is usually 00400000, which means that offset for code address 004D316F is D316F - because 00400000 + 000D316F = 004D316F. Sometimes this value represents offset from start of code section. Code section itself has offset of $1000, so in this case - code offset D216F means 00400000 + 00001000 + 000D316F = 004D316F.
- **Unit name**. Those are values like 'Project1' or 'Project1.dpr', 'Unit1.pas' and so forth. It's the name of Delphi or C++ Builder unit, which contains code, associated with code address. Can be '' (empty string), if there is no debug information available to set "code address" <-> "pas source" correspondence.
- **Class name**. Those are values like 'TForm1' or 'TMyClass'. It's name of class, which contains method with code address. Can be empty if there is no debug information available or if code address belongs to usual function or procedure (not to method). Sometimes can be combined with next item.
- **Method or routine name**. Those are values like 'Button1Click', 'DoSomething', etc. It's the name of method, procedure or function, which contains code address. Can be empty, if there is no debug information.
- **Line number**. Those are values like 39, 458, 31958, etc. It's # of line in .pas/.dpr/.cpp file, which compiles into code address. Can be 0, if there is no debug information.
- **Line number offset**. Those values like +0, +1, +5, etc. It's difference (usually in DEC) between line number and first line in current method (in lines). For example, if code address points to third line in procedure DoSomething, which starts at line 89, then line number will be 92 and line number offset will be +3 (since 89 + 3 = 92). There can be +/-1 line difference, depending on how lines are calculated. A detailed explanation on offsets can be found in this article 578.
- **Code offset from procedure's start**. Those values like +16, + $5 or +$0. Usually it's in HEX format. It's difference between code address and address of first line in current

method (in bytes).

See next part 83 to know how to extract these part of information from various formats of call stacks or skip to article about searching bug's location 95.

See also:

- Multi-threaded call stacks 85
- Configuring call stack 48
- Reading and understanding bug reports 72
- Call stacks 79
- Call stack formats 83
- Searching bug's location 95
- Final notes on call stack 96
- Searching bug's reason 97

7.1.2.2.1 Call stack formats

This article is a part of working with bug reports 72.

Here is how you can extract call stack information 81 from various formats of call stacks.

Delphi

- **Code address**. It's a first part of the line in form ':**7796d0e9**' (starting with ':'). This value is usually missed - unless line represents code without debug information (like system DLLs, for example).
- **Stack address**. Not specified.
- **Executable module name**. If code address is specified, then this is value right after code address - for example ':7796d0e9 **kernel32**'. If code address is not specified, then this value is omitted.
- **Code offset**. Not specified.
- **Unit name**. First word in line, if code address is not present. For example, '**Forms**'. If code address is present, then this value is skipped.
- **Class name**. Follows unit name after '.'. For example, 'Forms.**TCustomForm**'.
- **Method or routine name**. If code address is present - follows executable module name. For example, ':7796d0e9 kernel32.**BaseThreadInitThunk**'. Otherwise it follows class name (if present) or unit name (if class name is not present). For example, 'Forms.TCustomForm.**Create**'.
- **Line number**. Not specified.
- **Line number offset**. Not specified.
- **Code offset from procedure's start**. If code address is specified, then this is last value in form: '+ 0x12'. For example, ':7796d0e9 kernel32.BaseThreadInitThunk+ **0x12**'. If code address is not specified, then this value is not present.

EurekaLog (raw text)

- **Code address**. It's a value in 'Address' column.
- **Stack address**. It's a value in 'Stack' column.
- **Executable module name**. It's a value in 'Module' column.
- **Code offset**. It's a value in 'Offset' column. It's offset from module start (i.e. including $1000 offset for code's section).
- **Unit name**. It's a value in 'Unit' column.
- **Class name**. It's a value in 'Class' column.
- **Method or routine name**. It's a value in 'Procedure/method' column.
- **Line number**. It's first number in 'Line' column. For example, '**33**[3]'.
- **Line number offset**. It's number inside [] in 'Line' column. For example, '33[**3**]'.
- **Code offset from procedure's start**. Not specified.

EurekaLog (Viewer)

- **Code address**. It's a value in 'Address' column.
- **Stack address**. It's a value in 'Stack' column.

- **Executable module name**. It's a value in 'Module' column.
- **Code offset**. It's a value in 'Offset' column. It's offset from module start (i.e. including $1000 offset for code's section).
- **Unit name**. It's a value in 'Unit' column.
- **Class name**. It's a value in 'Class' column.
- **Method or routine name**. It's a value in 'Procedure/method' column.
- **Line number**. It's a value in 'Line' column.
- **Line number offset**. It's a value in 'Rel.line' column.
- **Code offset from procedure's start**. Not specified.

FastMM

- **Code address**. It's a first number in line. For example, '**4C0913**'.
- **Stack address**. Not specified.
- **Executable module name**. Not specified.
- **Code offset**. Not specified.
- **Unit name**. Unit file name is a first word inside [], follows code address. May be skipped if not available. For example, '4C0913 [**Forms**]'. Unit name is also a first word inside next [] pair. For example, '4C0913 [Forms][**Forms**.TCustomForm.AfterConstruction]'.
- **Class name**. Follows unit name after '.'. For example, '4C0913 [Forms][Forms.**TCustomForm**.AfterConstruction]'.
- **Method or routine name**. Follows class name (if present) or unit name (if class name is not present, but unit name is present). '.'-separated. For example, '4C0913 [Forms][Forms.TCustomForm.**AfterConstruction**]'. May also follow code address, if unit name is not present. For example, '75B94911 [**BaseThreadInitThunk**]'.
- **Line number**. Number inside last's []. For example, '4CC5DC [Unit15.pas][Unit15][Unit15.TForm15.FormCreate][**43**]'.
- **Line number offset**. Not specified.
- **Code offset from procedure's start**. Not specified.

JCL

- **Code address**. It's a first number in []. Usually follows executable module name. For example, '(00087ABC){Project1.exe} [**00488ABC**]'.
- **Stack address**. Not specified.
- **Executable module name**. It's a first word in {}. Follows code offset, if it's present. Otherwise it's a first value in the line. For example, '(00087ABC){**Project1.exe**}'.
- **Code offset**. First value in line, if present. It's inside (). For example, '(**00087ABC**)'. This is offset from start of code section (i.e. it doesn't include $1000 offset).
- **Unit name**. First word in line without any kind brackets. For example, '(00087ABC){Project1.exe} [00488ABC] **Unit1**'.
- **Class name**. Follows unit name after '.'. For example, '(00055523){Project1.exe} [00456523] Controls.**TWinControl**'.
- **Method or routine name**. Follows class name (if present) or unit name (if class name is not present, but unit name is present). '.'-separated. For example, '(00087ABC){Project1.exe} [00488ABC] Unit1.**A**'.
- **Line number**. It's number after "Line" word. For example, '(00087ABC){Project1.exe} [00488ABC] Unit1.A (Line **44**)'
- **Line number offset**. Follows line number and unit file name. Has a form of '+ 1'. For example, '(00087ABC){Project1.exe} [00488ABC] Unit1.A (Line 44, "Unit1.pas" + **0**)'.
- **Code offset from procedure's start**. Follows ()-part with line number information. Has a form of '+ $6C'. For example, '(00087ABC){Project1.exe} [00488ABC] Unit1.A (Line 44, "Unit1.pas" + 0) + $**0**'.

madExcept

- **Code address**. It's a first part of the line. For example, '**0086ad33**'.
- **Stack address**. Not specified.
- **Executable module name**. It a first word in line. It follows code offset. Space-delimited from other parts. For example, '0086ad33 +001f **Project1.exe**'.
- **Code offset**. Not specified.
- **Unit name**. Second word in line, follows executable module name. For example,

'0086ad33 +001f Project1.exe **OleCtrls**'. May be skipped if not available.

- **Class name**. Follows line number information. Class and method names are last information in the line. For example, '0086ad33 +001f Project1.exe OleCtrls 640 +11 **TOleControl**.Destroy'.
- **Method or routine name**. Follows class name (if present) or line numbers section (if class name is not present, but unit name is present). For example, '0086ad33 +001f Project1.exe OleCtrls 640 +11 TOleControl.**Destroy**'.
- **Line number**. Number which follows unit name. For example, '0086ad33 +001f Project1.exe OleCtrls **640** +11'.
- **Line number offset**. It follows line number. Has a form '+16'. For example, '0086ad33 +001f Project1.exe OleCtrls 640 +**11**'.
- **Code offset from procedure's start**. It follows code address. Has a form '+0009'. For example, '0086ad33 +**001f**'.

Now, when you have all information, it's time to put it in use and try to find bug's location 95.

See also:

- Multi-threaded call stacks 85
- Reading and understanding bug reports 72
- Call stacks 79
- How to read call stack 81
- Searching bug's location 95
- Final notes on call stack 96
- Searching bug's reason 97

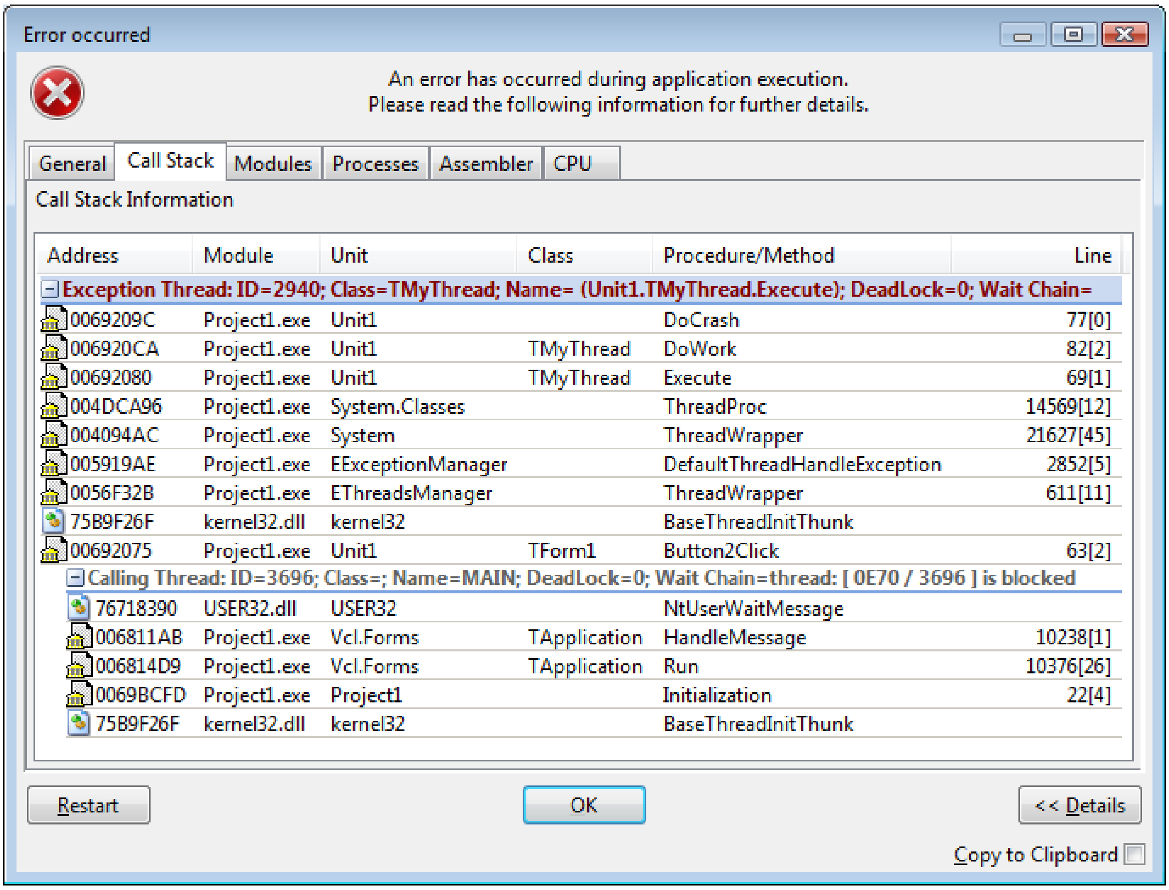
7.1.2.2.2 Multi-threaded call stacks

This article is a part of working with bug reports 72.

Important note: please read Using EurekaLog in multi-threaded applications 547 article before reading this article.

Call stacks in multi-threaded applications contain additional information.

For example:



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=2940; Class=TMyThread; Name= (Unit1.TMyThread.Execute); DeadLock=0; Wait Chain=					
0069209C	Project1.exe	Unit1		DoCrash	77[0]
006920CA	Project1.exe	Unit1	TMyThread	DoWork	82[2]
00692080	Project1.exe	Unit1	TMyThread	Execute	69[1]
004DCA96	Project1.exe	System.Classes		ThreadProc	14569[12]
004094AC	Project1.exe	System		ThreadWrapper	21627[45]
005919AE	Project1.exe	EExceptionManager		DefaultThreadHandleException	2852[5]
0056F32B	Project1.exe	EThreadsManager		ThreadWrapper	611[11]
75B9F26F	kernel32.dll	kernel32		BaseThreadInitThunk	
00692075	Project1.exe	Unit1	TForm1	Button2Click	63[2]
⊟ Calling Thread: ID=3696; Class=; Name=MAIN; DeadLock=0; Wait Chain=thread: [0E70 / 3696] is blocked					
76718390	USER32.dll	USER32		NtUserWaitMessage	
006811AB	Project1.exe	Vcl.Forms	TApplication	HandleMessage	10238[1]
006814D9	Project1.exe	Vcl.Forms	TApplication	Run	10376[26]
0069BCFD	Project1.exe	Project1		Initialization	22[4]
75B9F26F	kernel32.dll	kernel32		BaseThreadInitThunk	

Exception in background thread (GUI)

```
Call Stack Information:
-----------------------------------------------------------------------------------
-----------
|Address |Module      |Unit             |Class       |Procedure/Method            |
Line      |
-----------------------------------------------------------------------------------
-----------
|*Exception Thread: ID=2940; Parent=3696; Priority=0
          |
|Class=TMyThread; Name= (Unit1.TMyThread.Execute)
          |
|DeadLock=0; Wait Chain=
          |
|Comment=
          |
|----------------------------------------------------------------------------------
----------|
|0069209C|Project1.exe|Unit1             |            |DoCrash                     |
77[0]     |
|006920CA|Project1.exe|Unit1             |TMyThread   |DoWork                      |
82[2]     |
|00692080|Project1.exe|Unit1             |TMyThread   |Execute                     |
69[1]     |
|004DCA96|Project1.exe|System.Classes    |            |ThreadProc                  |
14569[12] |
|004094AC|Project1.exe|System            |            |ThreadWrapper               |
21627[45] |
|005919AE|Project1.exe|EExceptionManager|            |DefaultThreadHandleException|
2852[5]   |
|0056F32B|Project1.exe|EThreadsManager  |            |ThreadWrapper               |
611[11]   |
|75B9F26F|kernel32.dll|kernel32         |            |BaseThreadInitThunk         |
         |
|00692075|Project1.exe|Unit1            |TForm1      |Button2Click                |
63[2]     |
|----------------------------------------------------------------------------------
----------|
|Calling Thread: ID=3696; Parent=0; Priority=0
          |
|Class=; Name=MAIN
          |
|DeadLock=0; Wait Chain=thread: [ 0E70 / 3696 ] is blocked
          |
|Comment=
          |
|----------------------------------------------------------------------------------
----------|
|76718390|USER32.dll  |USER32           |            |NtUserWaitMessage           |
         |
|006811AB|Project1.exe|Vcl.Forms        |TApplication|HandleMessage               |
10238[1]  |
|006814D9|Project1.exe|Vcl.Forms        |TApplication|Run                         |
10376[26] |
|0069BCFD|Project1.exe|Project1         |            |Initialization              |
22[4]     |
|75B9F26F|kernel32.dll|kernel32         |            |BaseThreadInitThunk         |
         |
-----------------------------------------------------------------------------------
-----------
```

Exception in background thread (text)

It contains 2 call stacks: one per each thread. There are two threads in this application: main thread and a background thread based on TThread class. There was exception raised in the background thread.

This example illustrates several multi-threaded features of EurekaLog:

Call stacks for multiple threads

Bug report contains a single call stack by default. This call stack is created for thread which raised the exception. Such thread is called "Exception thread". This is the only thread in single-threaded application (it will be main thread for single-threaded application). Sometimes you may be interested in other threads, you may want to know what other threads are doing when exception had occurred. You can capture call stacks of other threads in application by checking "Capture stack of RTL threads" or "Capture stack of Windows threads" options 246.

Notes:

- there may be multiple call stacks for leak reports. There will be one call stack per each found leak. This behavior is customizable here 250. This behavior is not related to multi-threading;
- capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed.

Different call stacks types

EurekaLog marks call stacks as:

- **Exception thread**. This is thread which raised the exception. There can be only one exception thread in any bug report. This thread is listed first. It is mandatory; any other threads are optional;

- **Calling thread**. This is parent thread for exception thread. Normally, there is only one calling thread in bug report. There may be more than one calling thread if exception occurred inside synchronized routine (see below). This thread follows exception thread in the list. It's optional. It will be not shown if this thread was terminated before exception or if the corresponding "Capture stack of *XYZ* threads" option is turned off.
- **Running thread**. Any other thread is called "running thread". Main thread can also be a running thread. All running threads are listed after exception thread and calling thread(s) (if any).

Note: the meaning of exception and calling threads are changed significantly for exceptions within synchronized methods (see below).

Different thread types

EurekaLog can recognize the following thread types:

- RTL theads. These are threads created by Delphi/C++ Builder code. Such threads are created via:
  o TThread 553 or TThreadEx 559 class;
  o BeginThread 549 or BeginThreadEx 551 function.
- Windows threads. Any other threads are considered as "Windows threads". Such threads are created via CreateThread function.

It's recommended to create your own threads via TThreadEx 559 class or via BeginThreadEx 551 function (or via any other function which use TThread 553 or BeginThread 549 to create threads). Do not use CreateThread function to create threads. This will allow you to distinguish between your threads and system threads (system or 3rd party code may create additional threads in your application for background/service tasks).

- If you're using TThread(Ex) approach - then EurekaLog will be able to extract (child) class name and show it in the call stack's header as "Class=TMyThread".
- If you're using BeginThread(Ex) or CreateThread approach - then EurekaLog will be able to extract thread function and show it in the call stack's header as "Name= (ThreadProc)". Thread function's name will be printed in parentheses (see "Naming threads" below to know more about "Name" parameter). Thread function's name will always be Execute method for TThread approach.

Naming threads

You can name threads to simplify debugging. Since you can run multiple threads with the same thread class (or thread function) - this means that you can not distinguish between these threads by using only class name (or thread function's name). You need some additional marker. Such marker is a thread name.

You can supply thread name directly to TThreadEx 559 class or BeginThreadEx 551 function:

```
uses
  EBase;

  TH := BeginThread(nil, 0, MyThreadProc, Args, 0, TID, 'This is my thread');

// or:

type
  TMyThread = class(TThreadEx)
  // ...
  protected
    procedure Execute;
  // ...
  end;

procedure TMyThread.Execute;
begin
  inherited;
  // ...
end;

  Thread := TMyThread.Create(False, 'This is my thread');
```

If you are not using TThreadEx/BeginThreadEx - then you can name a thread from the thread itself. You can set thread's name by using NameThread function from EThreadsManager unit. You should call this function as a very first action inside thread function (for BeginThread) or Execute method (for TThread). Thread name can be arbitrary string, so you can append any parameters that you need to distinguish between threads. For example:

```
{$IFDEF EUREKALOG}
uses
  EThreadsManager;
{$ENDIF}

type
  PThreadArguments = ^TThreadArguments;
  TThreadArguments = record
    FileName: String;
    // ... any other arguments
  end;

function MyThreadProc(Parameter: Pointer): Integer;
var
  Args: TThreadArguments;
begin
  Args := PThreadArguments(Parameter)^;
  Dispose(PThreadArguments(Parameter));

  {$IFDEF EUREKALOG}NameThread('FileName=' + Args.FileName);{$ENDIF}

  // ... actual thread's code goes there
end;

...

var
  Args: PThreadArguments;
  TID: Cardinal;
  TH: THandle;
begin
  Args := AllocMem(SizeOf(TThreadArguments));

  Args.FileName := Edit1.Text;

  TH := BeginThread(nil, 0, MyThreadProc, Args, 0, TID);

  if TH = 0 then
    Dispose(Args)
  else
    CloseHandle(TH);
end;
```

Any thread name will be shown in the call stack's header as "Name=*your-thread-name*" (without parentheses; part inside parentheses is a name of thread function - see above). For the example (the one directly above) it will be:

```
   Name=FileName=C:\Sample.txt (MyThreadProc)
```

(assuming that Edit1.Text holds 'C:\Sample.txt' string)

Note: main thread will always have a name "MAIN".

Indicating creator of the thread

Since you can run multiple threads with the same thread class (or thread function) **and** with the same arguments from multiple locations - this means that you can not distinguish between these threads by using class name (or thread function's name) and any properties of the thread itself. You need to use name of thread's creator.

EurekaLog appends creator of the thread to the end of thread's call stack. Typically this is a single entry below "BaseThreadInitThunk" system function. For example, this example of the call stack shows that TMyThread was created in Unit1.TForm1.Button2Click method, line 63:

```
Call Stack Information:
------------------------------------------------------------------------------------
-----------
|Address |Module      |Unit                  |Class        |Procedure/Method                |
Line      |
-----------------------------------------------------------------------------------
-----------
|*Exception Thread: ID=2940; Parent=3696; Priority=0
          |
|Class=TMyThread; Name= (Unit1.TMyThread.Execute)
          |
|DeadLock=0; Wait Chain=
          |
|Comment=
          |
|----------------------------------------------------------------------------------
----------|
|0069209C|Project1.exe|Unit1           |            |DoCrash                     |
77[0]     |
 <-
...................................................................................
....->
|75B9F26F|kernel32.dll|kernel32        |            |BaseThreadInitThunk         |
         |
|00692075|Project1.exe|Unit1           |TForm1      |Button2Click                |
63[2]     |
|----------------------------------------------------------------------------------
----------|
```

Notes:

- call stack of the creator thread is not captured by EurekaLog, only single entry (immediate caller) is captured;
- call stack of the calling thread is not the call stack for the creator. Do not confuse these call stacks. Calling thread indicate state of creator's thread after thread creation had occurred;
- do not confuse last line in thread's call stack ("creator") as belonging to the thread itself. This code was executed by different thread (indicated by "Parent"). This line was not executed by the thread itself.

Indicating threads options

EurekaLog lists some options of threads, such as:

- ID value contains TID (Thread ID). This is system value that unique identifies threads within the system. Do not confuse TID with thread's handle - those are totally different things. TID (together with PTID - see below) can be used to check relations between threads;
- Parent value contain PTID (Parent Thread ID). This is ID of thread which created this thread. This value is always 0 for the main thread. Please note that thread identified by PTID may be already finished and terminated when exception had occurred;
- Priority value indicate thread's priority as returned by GetThreadPriority function.

Wait Chain Traversal feature support

EurekaLog has support for WCT (Wait Chain Traversal) feature available since Windows Vista. WCT is a mechanism for debugging blocked threads and processes, and detecting deadlocks.

Using WCT, debugging software can analyze executing processes and report on the state of threads, including information such as what a blocked thread is waiting for and whether a deadlock condition exists. Debugging software can analyze and then reports the state of threads. A thread's state (unblocked, blocked, or deadlocked) is reported as a wait chain. A wait chain is an alternating directed graph of threads and synchronization objects. In the graph, an edge from a thread to an object indicates that the thread is waiting for the object; an edge from an object to a thread indicates that the thread is the current owner of the object. For example, the following wait chain represents a thread (Thread A) that is blocked waiting for a mutex object that is owned by Thread B.

Thread A -> Mutex 1 -> Thread B

As illustrated here, the first node in a wait chain is the thread being analyzed and the remaining nodes are the elements (synchronization objects, processes, threads, and so on), if any, that are directly or indirectly causing the thread to block. The simplest wait chain reflects a thread that is not blocked. It is composed of a single node, representing the unblocked thread. A blocked thread is represented by a wait chain containing multiple nodes that is non-cyclic: that is, there are no two nodes in the chain that represent the same thread or object. When a thread is deadlocked, the wait chain that represents it is cyclic. Consider the scenario where Thread A owns Object 1 and is blocked waiting for Object 2, while Thread B owns Object 2 and is blocked waiting for Object 1. The wait chain is as follows:

Thread A -> Object 2 -> Thread B -> Object 1 -> Thread A

Note that Thread A appears twice. The second node for Thread A could be replaced by an edge (->) that connects Object 1 to the first Thread A node, creating a loop that represents the cyclic dependency.

- EurekaLog shows 1 in "Deadlock" value if deadlock (cyclic dependency) was detected for this thread. Otherwise "Deadlock" value is 0;
- EurekaLog shows thread's wait chain in "Wait chain" value. Simplest wait chains (that is wait chain which consists of a single node) are not shown ("Wait chain" value will be empty).

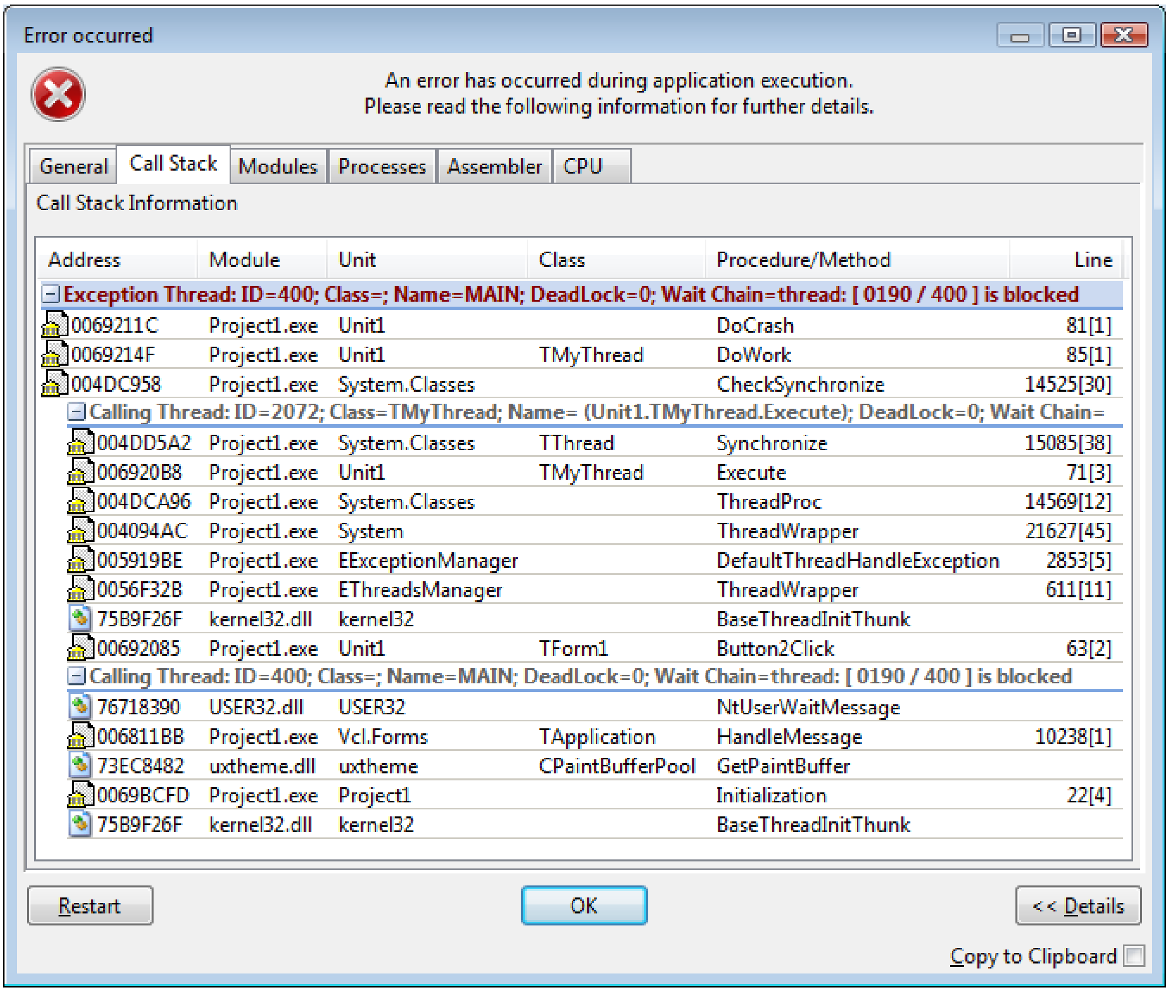
WCT feature is accessible only on Windows Vista and above. "Deadlock" and "Wait chain" values will be empty on older OS versions.

Parenting threads

EurekaLog lists TID ("ID") and PTID ("Parent") values to help you identify child/parent threads. Additionally, a parent thread for exception thread is marked as "Calling thread" and is shown below exception thread for your convenience. It will be not shown if this thread was terminated before exception or if the corresponding "Capture stack of *XYZ* threads" option is turned off.

Merging call stacks for Synchronize

EurekaLog will merge call stacks of main thread and invoking thread (that is the thread which called Synchronize method) when possible. The result may look like this:



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=400; Class=; Name=MAIN; DeadLock=0; Wait Chain=thread: [0190 / 400] is blocked					
0069211C	Project1.exe	Unit1		DoCrash	81[1]
0069214F	Project1.exe	Unit1	TMyThread	DoWork	85[1]
004DC958	Project1.exe	System.Classes		CheckSynchronize	14525[30]
⊟ Calling Thread: ID=2072; Class=TMyThread; Name= (Unit1.TMyThread.Execute); DeadLock=0; Wait Chain=					
004DD5A2	Project1.exe	System.Classes	TThread	Synchronize	15085[38]
006920B8	Project1.exe	Unit1	TMyThread	Execute	71[3]
004DCA96	Project1.exe	System.Classes		ThreadProc	14569[12]
004094AC	Project1.exe	System		ThreadWrapper	21627[45]
005919BE	Project1.exe	EExceptionManager		DefaultThreadHandleException	2853[5]
0056F32B	Project1.exe	EThreadsManager		ThreadWrapper	611[11]
75B9F26F	kernel32.dll	kernel32		BaseThreadInitThunk	
00692085	Project1.exe	Unit1	TForm1	Button2Click	63[2]
⊟ Calling Thread: ID=400; Class=; Name=MAIN; DeadLock=0; Wait Chain=thread: [0190 / 400] is blocked					
76718390	USER32.dll	USER32		NtUserWaitMessage	
006811BB	Project1.exe	Vcl.Forms	TApplication	HandleMessage	10238[1]
73EC8482	uxtheme.dll	uxtheme	CPaintBufferPool	GetPaintBuffer	
0069BCFD	Project1.exe	Project1		Initialization	22[4]
75B9F26F	kernel32.dll	kernel32		BaseThreadInitThunk	

Exception inside synchronized routine (GUI)

```
Call Stack Information:
-----------------------------------------------------------------------------------
---------------
|Address |Module      |Unit             |Class           |Procedure/Method
   |Line      |
-----------------------------------------------------------------------------------
---------------
|*Exception Thread: ID=400; Parent=0; Priority=0
              |
|Class=; Name=MAIN
              |
|DeadLock=0; Wait Chain=thread: [ 0190 / 400 ] is blocked
              |
|Comment=
              |
|----------------------------------------------------------------------------------
--------------|
|0069211C|Project1.exe|Unit1             |                |DoCrash
   |81[1]     |
|0069214F|Project1.exe|Unit1             |TMyThread       |DoWork
   |85[1]     |
|004DC958|Project1.exe|System.Classes    |                |CheckSynchronize
   |14525[30] |
|----------------------------------------------------------------------------------
--------------|
|Calling Thread: ID=2072; Parent=400; Priority=0
              |
|Class=TMyThread; Name= (Unit1.TMyThread.Execute)
              |
|DeadLock=0; Wait Chain=
              |
|Comment=
              |
|----------------------------------------------------------------------------------
--------------|
|004DD5A2|Project1.exe|System.Classes   |TThread         |Synchronize
   |15085[38] |
|006920B8|Project1.exe|Unit1            |TMyThread       |Execute
   |71[3]     |
|004DCA96|Project1.exe|System.Classes   |                |ThreadProc
   |14569[12] |
|004094AC|Project1.exe|System           |                |ThreadWrapper
   |21627[45] |
|005919BE|Project1.exe|EExceptionManager|                                          |
DefaultThreadHandleException|2853[5]   |
|0056F32B|Project1.exe|EThreadsManager  |                |ThreadWrapper
   |611[11]   |
|75B9F26F|kernel32.dll|kernel32         |                 |BaseThreadInitThunk
   |          |
|00692085|Project1.exe|Unit1            |TForm1          |Button2Click
   |63[2]     |
|----------------------------------------------------------------------------------
--------------|
|Calling Thread: ID=400; Parent=0; Priority=0
              |
|Class=; Name=MAIN
              |
|DeadLock=0; Wait Chain=thread: [ 0190 / 400 ] is blocked
              |
|Comment=
              |
|----------------------------------------------------------------------------------
--------------|
|76718390|USER32.dll  |USER32           |                 |NtUserWaitMessage
   |          |
|006811AB|Project1.exe|Vcl.Forms        |TApplication    |HandleMessage
   |10238[1]  |
|006814D9|Project1.exe|Vcl.Forms        |TApplication    |Run
   |10376[26] |
|0069BCFD|Project1.exe|Project1         |                |Initialization
   |22[4]     |
|75B9F26F|kernel32.dll|kernel32         |                 |BaseThreadInitThunk
   |          |
-----------------------------------------------------------------------------------
---------------
```

Same exception in text

The above example should be translated as:

- there are two threads in application: main thread (ID = 400) and background/worker thread (ID = 2072). Although there are 3 call stacks in the report, but there are only two unique threads (as indicated by IDs). There are two call stacks for main thread (ID = 400): first call stack is exception thread and it was created inside Synchronize call, second call stack is usual call stack for main thread;
- worker thread (ID = 2072) was launched from Unit1.TForm1.Button2Click line 63[2]. This is indicated by last line in call stack for thread #2072 - right below system's BaseThreadInitThunk. Main thread does not have creator - this is indicated by

BaseThreadInitThunk being the last in the second call stack for the main thread (ID = 400);
- worker thread (ID = 2072) called `Synchronize` method. This is indicated by presence of `TThread.Synchronize` at the top of the call stack for thread #2072;
- task from `Synchronize` method was executed by main thread (ID = 400). Task is `DoWork` method from `TMyThead` class. This is indicated by `TThread.DoWork` above `CheckSynchronize` call for main thread (ID = 400);
- there was exception raised in `DoCrash` routine which was called from `TMyThread.DoWork`. This is indicated by exception call stack - which is the first call stack for main thread (ID = 400);
- main thread waits for window message (i.e. it is processing messages; it's inside message loop). This is indicated by last call stack for main thread (ID = 400).

Notes:
- First call stack for main thread ("exception thread") does not represent any *real-time* information. Main thread is doing message pumping at the current moment - as indicated by the second call stack for main thread. Exception thread shows some *older* state of main thread - that is the state when main thread was doing tasks for `Synchronize`. In other words, there are two call stacks for the single thread (main thread): one represents the past and second represents the current moment.
- "Exception thread" is not a real exception thread. The handled exception was (re-)raised by `TThread.S`
- Call stacks for manual synchronization (such as `SendMessage` into main thread) will not be merged.
- This feature supports merging call stacks for any exceptions that was re-raised into another thread. `Synchronize` calls is just a sub-case of generic cross-thread exceptions.
- This feature supports any TThread-descendant - such as AsyncCalls or OmniThreadLibrary.

See also:
- Using EurekaLog in multi-threaded applications 547
- Multi-threadeding options 246

7.1.2.3 Searching bug's location

This article is a part of working with bug reports 72.

1. When GUI is available

If you're working in Delphi, browse EurekaLog error dialog or use a EurekaLog viewer - you can just **double click** on any line in call stack. You'll be moved to that location. Very simple.

Few notes here:
1. A compiled application must match source files. Otherwise you will be moved to wrong location. What you can do to minimize chances for this to happen:
   - Open target unit in IDE. That's if you have multiple versions of the same unit.
   - EurekaLog is capable of using your __history folder, but in some cases you may need to extract files manually.
   - You may need to restore older version of file from SVN.
2. IDE must be launched and source files must be available (either opened or could be found by IDE).
3. IDE and application (error dialog or EurekaLog Viewer) must reside under the same security boundary (same user account and same elevation level under UAC).

2. When line number is available

When GUI doesn't work or not available, but you have line numbers in call stack - you need to open unit in IDE manually and use **Search / Go to line number** command in IDE main menu. Enter line number and press OK - you'll be moved to line location.

If you can't find a unit version, which matches your compiled application, then you may try to use **line number offset**. Open unit and find target procedure or method. Move down manually for line number offset lines. That will be your location.

3. When procedure/method name is available

If you don't have line numbers information or this information is obsolete (doesn't match source), but you have a subroutine name - you can find subroutine in target unit and try to guess, where the call was. For example, if you have such call stack

- DoSomething
- DoWork
- Run

And your DoWork routine looks like this:

```
procedure DoWork;
var
  X: Integer;
begin
  Prepare;
  DoSomething(-1);
  for X := 1 to 5 do
    DoSomething(X);
  Done;
end;
```

Then you can guess that call stack either points to first call `DoSomething(-1)` or second call `DoSomething(X)`. But it doesn't point to `Prepare` or `Done`.

5. When code address is available

Next fallback case is raw code address. There can be two options: either you have a code address OR you have module name and code offset.

Note: you don't need a base address of the module.

In either case you need to run IDE, open your project, run it and put it on pause (**Run** / **Pause**). Now, use the **Search** / **Go to address** command. Enter absolute code address. Don't forget to add '$', if you enter HEX.

Note: if you have code offset and module name, you need to calculate code address first. Go to **View** / **Debug windows** / **Modules** to open Modules window. Find your module here and get its base address. Now add a code offset to this address (include a $1000, if needed) - this will be your final code address.

When you press OK - you'll be moved to exact location. If it's possible - you'll see a source code (.pas file). If not (say, no debug information available) - you'll see a CPU debugger.

Second case (having a code offset + module name) is better than first case (having code address). Even though you have to make calculations first, but first case doesn't allow you to find location, if module on your machine is loaded at different location than module on client's machine.

See also:

- Reading and understanding bug reports 72
- Call stacks 79
- How to read call stack 81
- Call stack formats 83
- Final notes on call stack 96
- Searching bug's reason 97

7.1.2.4 Final notes on call stack

This article is a part of working with bug reports 72.

Here are few moments, which you should have in mind, when you are going to read call stacks.

First: one bug-report file may contain several bug-reports, and one bug-report may contain several call stacks (for example: one call stack for each thread). So, before actually reading – make sure, that you're going to read the required ("the one") call stack.

Usually, call stacks should be read from the bottom to the top, if you want to go in execution order (that is: from top to bottom, if you rather want to "unwind" from error's location): the first called routine is placed at their bottom. The next called routine is above it, on the next line. And so on. A current location (the last called routine) is located at the top of call stack (call stack's first line). Sometimes current location indicate the location of the problem (but it is not always so – see searching bug's reason 97), and the rest of the call stack indicate execution path – i.e. the way, how we got to the current place.

First few routines (from bottom) usually are system's ones or belongs to Delphi's RTL and, therefore, aren't very interesting. Often there are only partial information available for them – due to partial or missing debug information. Examples are things like 'BaseThreadInitThunk', 'Controls.TWinControl.WndProc' and so on. And sometimes the call stack itself has a limited depth (length/capacity) – i.e. it can not contain more than N lines or M characters. In that case the bottom part is simply cut.

On the other side: top part of the call stack may mention some RTL tool routines along with error's location and your code. That depends on many factors. For example, it depends on stage (level of processing), at which exception was caught and the call stack was created. The examples of such tool routines are RTL's exception processing helpers (like KiUserExceptionDispatcher), GetMem's calls, etc, etc.

Some tools also may specially add an additional line to the very top of the call stack. This line represents the exact error's location, if it is available (for example, it is an code instruction's address for exceptions). If this is the case, then if call stack already contains this address as its first line, then nothing happens. But if there are lines with RTL's tool routines at the top – then this address will be duplicated in the call stack. First time it'll be listed in the very first line (as special added) and the second time it'll be listed few lines below in the call stack (after RTL's tool routines).

The next thing: you should know that on x86-32 **there is no reliable way to build exact call stack**. All methods use some kind of assumptions, heuristics and guessing. And sometimes methods may be confused, so you may have a false-positive entries in call stack or missed entries. See also: different stack tracing methods 578.

So, if you're analyzing the call stack and see strange thing ("Hey? What the…? This routine can not be called here by my code!") – you should take this into account and suppose that: either there is call missed or there is unnecessary (false-positive) call. You should also remember about this difference, if you use call stack in your application directly (for example, to get textual description of your caller).

Now, if you've read and understand this - you may call yourself an expert with call stacks (well, a bit of a practice will not hurt).

See also:

- Reading and understanding bug reports 72
- Call stacks 79
- Stack tracing: RAW method and frame-based method 578
- How to read call stack 81
- Call stack formats 83
- Searching bug's location 95
- Searching bug's reason 97

7.1.2.5 Searching bug's reason

This article is a part of working with bug reports 72.

Following call stack... or not?

In many cases the reason for the problem may be obvious - all you need to do is to follow a call stack 95 in the bug report and you get your buggy code. The only thing left in to analyze variables in that lines, check assumptions, conditions and figure how to fix it.

However, not always call stack points you to a problem. This is especially true for the Access Violation exception 160. For example - suppose that you have a memory-corruption bug in your application. Buggy code may run without a visible problem and just corrupt some other memory. And the actual exception will be raised later, when fully unrelated code will be executed. Now you have Access Violation and call stack, which points to some innocent code, which have nothing to do with the original problem.

You should just mindfully analyze the situation and do not blame unrelated code.

Some types of bug-reports can not point to the problem by definition. For example - memory problems (leaks and corruption). Memory manager just can not scan the entire memory pool for problems at every machine instruction in your application. What's more: it is not possible to scan entire pool on every request to memory manager either. As there is a lot of used memory in every application. So, if you'll scan all memory at every call to memory manager, then your performance will be near zero.

That's why memory manager usually checks only blocks, which are directly related to the current operation. For example, when we ask to free memory, then memory manager will check only this block for corruption. Note: it'll check only this block, not all allocated blocks. Next example: we're asking for memory. So, memory manager will go through available "free" blocks and pick suitable one. Before returning it to us, memory manager will scan it to check, that no one have wrote into that block while it was free. See how to solve memory problems 171.

That is why problems with memory will be raised later, and not in the moment, when they actually occurred.

Another example is memory leak 166.

Using other information in the bug report

Apart from call stack - there may be other information available in the bug reports. You can extract hints to the problem from this auxiliary information. The most important are information about CPU register and memory dumps. Sometimes you may extract information about variables from these pieces. Of course, it requires assembler knowledge, but it can be very handy sometimes.

Actually, the memory dumps can be useful even for un-experienced programmer. For example, the dump of memory leak can tell you, what data was there at run-time. Of, you can see, say, a string's data instead of object's layout and so you can trace where did this string go and find the source of memory corruption.

Unfortunately, it is really difficult to give some general advices here – almost every case require individual approach. When you suggest (guess) what the situation can be at run-time in your application - then the additional information in the report can help you to check this guess.

Well, an example. The very strange access violation at, seemingly, usual place at function's call. You may notice that your application was launched in Windows 2000 (that is field in the bug report). And you do know, that you didn't tested your application on this system. So, you dig a little deeper and discover, that used function do not exists in Windows 2000, and so this is the reason for the problem (say, you've imported the function via GetProcAddress, but didn't check for errors).

See also:

- Reading and understanding bug reports 72
- Call stacks 79
- How to read call stack 81

- Call stack formats 83
- Searching bug's location 95

7.1.3 Modules section

Modules section of bug report contain list of loaded executables (i.e. single host exe and all DLLs). This list can be disabled in options 266.

Note: modules list is automatically disabled for reporting leaks.

```
Modules Information:
------------------------------------------------------------------------------------
------------------------------------
|Handle  |Name        |Description                      |Version        |Size     |
Modified           |Path                |
------------------------------------------------------------------------------------
------------------------------------
|00400000|Project1.exe|                              |1.0.0.0       |16512079|2013-
08-17 19:15:26|C:\Projects         |
|76A90000|user32.dll  |Multi-User USER API Client DLL|6.0.6002.18005|820224  |2009-
04-11 20:22:43|C:\Windows\System32\|
|76CB0000|kernel32.dll|Windows NT BASE API Client DLL|6.0.6002.18704|1210368 |2012-
09-28 20:34:50|C:\Windows\System32\|
|76FF0000|ntdll.dll   |NT Layer DLL                  |6.0.6002.18881|1585256 |2013-
07-09 16:04:30|C:\Windows\System32\|
|77190000|psapi.dll   |Process Status Helper         |6.0.6001.18000|16896   |2008-
01-21 06:50:47|C:\Windows\System32\|
....
------------------------------------------------------------------------------------
------------------------------------
```

Example of a modules list (list is shortened)

The list contains the following columns:

- Handle value indicate base address of loaded module. This value can be used to calculate offsets (or convert offsets to absolute addresses). Entire list is sorted on that column. Host exe have base address of $400000. This value have 8 hex-characters for 32-bit bug reports and 16 hex-characters for 64-bit code;
- Name value indicate file name of loaded module without path;
- Description value shows module's description extracted from version information (if available). "File description" field is used as source for description;
- Version value shows module's version information (if available). dwFileVersion field from VS_FIXEDFILEINFO structure is used as source;
- Size value shows size of exe/DLL file in bytes;
- Modified value shows file's modification time in local time zone;
- Path value shows folder which contain loaded module. Path and name values form a full file name.

See also:

- Understanding bug reports 72
- Understanding call stacks 78

7.1.4 Processes

Processes section of bug report contain list of running processes. This list can be disabled in options 266.

Note: processes list is automatically disabled for reporting leaks.

```
Processes Information:
------------------------------------------------------------------------------------
-----------------------
```

```
|ID |Name            |Description          |Version       |Memory  |Priority|
Threads|Path                |
-----------------------------------------------------------------------------------
-----------------------
|0  |[System Process]|                     |              |0       |        |2
 |                    |
|4  |System          |                     |              |0       |Normal  |100
 |                    |
|320|TSVNCache.exe   |TortoiseSVN cache    |1.8.0.24401   |11210752|Normal  |11
 |C:\TortoiseSVN\bin\ |
|364|taskeng.exe     |Task Scheduler Engine|6.0.6002.18342|11407360|Normal  |11
 |C:\Windows\System32\|
|404|svchost.exe     |                     |              |0       |Normal  |20
 |                    |
|608|Project1.exe    |                     |1.0.0.0       |0       |Normal  |20
 |C:\Projects\        |
...
-----------------------------------------------------------------------------------
-----------------------
```

Example of a processes list (list is shortened)

The list contains the following columns:

- ID value shows Process ID (PID). PID is unique system value that identifies running process within the system. List is sorted by this column;
- Name value shows process name - that is a name of exe file. This value may be not available for some system processes. Please note that name of the process can be available, but full file name (i.e. path and name) and some other values can still be not available;
- Description value shows description extracted from version information of host exe for the process (if available). "File description" field is used as source for description. This value is typically missed for system processes, because current process have no access to open system processes for retrieving information;
- Version value shows version information of host exe (if available). dwFileVersion field from VS_FIXEDFILEINFO structure is used as source. This value is typically missed for system processes, because current process have no access to open system processes for retrieving information;
- Memory value indicate process working set size in bytes. Please note that this value is not virtual memory of the process. This value is typically missed for system processes, because current process have no access to open system processes for retrieving information;
- Priority value shows current process priority;
- Threads value shows number of running threads inside process;
- Path value shows folder which contain host exe. Path and name values form a full file name. This value is typically missed for system processes, because current process have no access to open system processes for retrieving information.

See also:

- Understanding bug reports 72
- Understanding call stacks 78

7.1.5 Assembler

Assembler section shows disassembler machine code near exception. This section can be disabled in options 266.

Note: assembler section is automatically disabled for reporting leaks.

Disassembler listing does not provide any new information as compared to other information in bug report - because it simply duplicate content of your executable, but presents it in human-readable form. But this section can still be useful, because it saves you time on disassembling your exe/DLL file.

```
Assembler Information:
--------------------------------------------------------------------------
; Base Address: $7F0000, Allocation Base: $400000, Region Size: 61440
; Allocation Protect: PAGE_EXECUTE_WRITECOPY, Protect: PAGE_EXECUTE_READ
; State: MEM_COMMIT, Type: MEM_IMAGE
;
;
; Unit9.TForm9.Button7Click (Line=747 - Offset=0)
; ----------------------------------------------
00000000007F0D20  55              PUSH RBP
00000000007F0D21  4883EC20        SUB  RSP, $20
00000000007F0D25  488BEC          MOV  RBP, RSP
00000000007F0D28  48894D30        MOV  [RBP+$30], RCX
00000000007F0D2C  48895538        MOV  [RBP+$38], RDX
;
; Line=748 - Offset=16
; --------------------
00000000007F0D30  488B0DA9ACC5FF  MOV  RCX, [REL -$003A5357] ; ($000000000044B9E0)
Exception Data as ANSI: '¨ºD'; Data as UNICODE: '모D'
00000000007F0D37  B201            MOV  DL, 1
00000000007F0D39  4C8D0520000000  LEA  R8, [REL $00000020]   ; ($00000000007F0D60)
Unit9.TForm9.Button7Click (Line=749) UNICODE: 'Error Message'
00000000007F0D40  E84BEBC6FF      CALL -$3914B5              ; ($000000000045F890)
System.Exception.Create
00000000007F0D45  4889C1          MOV  RCX, RAX
00000000007F0D48  E823C3C1FF      CALL -$3E3CDD              ; ($000000000040D070)
System._RaiseExcept <-- EXCEPTION
;
; Line=749 - Offset=45
; --------------------
00000000007F0D4D  488D6520        LEA  RSP, [RBP+$20]
00000000007F0D51  5D              POP  RBP
```

Example of disassembly output

Exception address will be listed in the middle of disassembly listing and marked with "<-- EXCEPTION" comment. Listing will go 15 CPU instructions in both directions (up/down) from exception address or until start/end of routine is found.

Note: exact value of exception address can be found in General 74 section or CPU 103 section.

Assembler section uses asm syntax compatible with Delphi's asm-blocks with minor exceptions (see below):

- Each line starts with absolute address of that CPU instruction inside the module. Address is represented as 8 hex-characters for 32-bit code or 16 hex-characters for 64-bit code;
- The next part is RAW dump which shows exact bytes of CPU instruction. These are 2 hex-characters per byte. Exact length of this block can vary depending on actual CPU instruction. For example, for x86-32 and x86-64 this can be from 2 hex-characters (1 byte) up to 30 hex-characters (15 bytes - which is maximum length of CPU instruction for x86). Raw bytes can be used to manually disassemble code* 103 (see remark below);
- Central piece is CPU instruction and (optional) arguments. Assembly syntax follows Delphi compiler with minor exceptions. Please note that instructions will use addresses and offsets instead of textual identifiers;
- Each line may end with optional comment block. This optional comment can show:
  - o Exception line mark. "<-- EXCEPTION". This mark indicate address of CPU instruction which raised the exception. This mark can be shown only once per entire listing;
  - o Absolute address. This value is shown if instruction references relative address and absolute address can be calculated. For example, MOV, JMP, and CALL instructions ("CALL -$3E3CDD ; ($000000000040D070)"). This value can also be shown for indirect values;
  - o Identifier. This value is shown if there is debug information available for that address.

For example: "`CALL -$3E3CDD ; System._RaiseExcept`". This value usually indicate routine's name and line for JMP and CALL instructions, but can also indicate code blocks for try/except statements. Please note that this value can show useless information for data references - if this data is stored inside code section, for example: "`LEA R8, [REL $00000020] ; Unit9.TForm9.Button7Click (Line=749)`";

- String data. This value is shown when instruction references AnsiString or UnicodeString constant. For example: "`LEA R8, [REL $00000020] ; UNICODE: 'Error Message'`". This value can not be false-positive, because EurekaLog analyzes string's header;
- Character data. This value is shown if instruction references some memory that can be interpreted as string (PAnsiChar and PWideChar). Note that this case does not include AnsiString and UnicodeString constants, because these were covered in previous item. This value is often false-positive because EurekaLog only checks for data to be readable and zero-terminated. Still, this value can be useful. For example, for MessageBox's text/captions. Example: "`LEA RDX, [REL $000000BD] ; Data as ANSI: 'E'; Data as UNICODE: 'Error'`". This value is always shows ANSI and Unicode interpretation of data. This value also shows references to code section, because PChar constants are often encoded right in code section.

CPU instructions are grouped in source lines. Each source line is marked with comment block above CPU instructions. Comment block indicate routine name, line number and byte offset. Each value is listed only if it was changed from previous line. For example, routine name is not shown if current line belongs to the same routine as previous line.

There is also a special comment block at the top of the section which shows general information about code page to which disassembled code belongs.

Note: disassembly listing uses ;-style comments instead of usual Delphi's //-style comments.

Absolute and relative addressing

EurekaLog's disassembler follows rules of Delphi's disassembler. For example, x86-32 assembler uses absolute addressing by default, but x86-64 assembler uses relative addressing by default. For example: "`MOV EAX, [$10]`". This instruction will be compiled into "`MOV EAX, [ABS $10]`" by 32-bit assembler. But the same instruction will be compiled into "`MOV EAX, [REL $10]`" (which means "`MOV EAX, [RIP + $10]`") by 64-bit assembler.

Delphi assembler supports ABS and REL modifier to alter addressing mode. It does not support [RIP + $10] or [$ + $10] syntax. ABS modifier is shown as + sign by Delphi's disassembler. For example, "`MOV EAX, [ABS $10]`" instruction will be disassembled by Delphi as "`MOV EAX, [+$10]`".

EurekaLog's disassembler always use REL and ABS modifiers to indicate addressing mode. REL/ABS modifiers are omitted for 32-bit code.

x86-64 is extension of x86-32. Thus, x86-64 tries to be compatible with x86-32 as much as possible. Still, there are differences - such as addressing mode. x86-32 prefers absolute addressing, while x86-64 prefers relative addressing to simplify writing PIC (Position Independent Code). This means that 64-bit code requires much less fix-ups when loading DLL to address which is different from preferred base address of this DLL. That's because 64-bit instructions works with relative addresses, so it doesn't use absolute addresses which needs to be fixed if code is loaded to another address.

Summary:

```
MOV [sym],       RAX  ; absolute for 32-bit, relative for 64-bit
MOV [REL sym],   RAX  ; relative
MOV [RIP + sym], RAX  ; relative, not supported by Delphi
MOV [$ + sym],   RAX  ; relative, not supported by Delphi
MOV [ABS sym],   RAX  ; absolute
MOV [+ sym],     RAX  ; absolute, Delphi disassembler only
```

64-bit values

Most 64-bit instructions works with 32-bit values. In 64-bit mode, immediates (that is, contants) and displacements (that is, offsets) are generally only 32 bits wide.

1. The only instruction which takes a full 64-bit immediate is: "`MOV reg64, imm64`". For example: "`MOV RBX, $1234567890ABCDEF`". Any other form of immediate will be 32-bit only (zero-extended to 64-bit prior to use, if needed).
2. The only instructions which take a full 64-bit displacement is loading or storing, using MOV, AL, AX, EAX or RAX (but **no other registers**) to an absolute 64-bit address. For example: "`MOV RAX, [ABS $1234567890ABCDEF]`". Address must be 64-bit absolute address (no registers are allowed in the effective address, and the address cannot be RIP-relative). Recipient register must be some form of AX register. Any other form of displacement will be 32-bit only (sign-extended prior to use, if needed).

The above means that you can not JMP or CALL to 64-bit absolute location specified with constant. JMP and CALL can only jump to a relative location within +/- 2 Gb from current instruction. If you need to jump to arbitrary 64-bit location then you have to use at least two instructions or indirect addressing:

```
MOV RAX, $1234567890ABCDEF
JMP RAX
```

or:

```
JMP [REL 0]
DB $12, $34, $56, $78, $90, $AB, $CD, $EF
```

(*) **Note:** automatic disassembling may be not possible in the following cases:

- EurekaLog's disassembler may encounter unknown instruction's opcode. Currently there is support for Intel's and AMD's opcodes up to SSE 4.1 (including AMD's 3DNow opcodes). Newer opcodes will be not recognized. Currently there is no support for VEX and XOP opcodes (except for a few). There is no support for 3rd party opcodes (such as special virtual machine opcodes);
- x86 architecture does not allow you to "move backward". I.e. you can't get 100% correct address of previous CPU instruction by having address of some valid CPU instruction. This makes 100% correct disassembling of code BEFORE exception address impossible. EurekaLog uses educated guesses (such as debug information or trying to walk forward from some address before exception), but these guesses may sometimes fail. This means that while exception address and listing below it will always be 100% accurate, but listing above exception address may show wrong values.

See also:

- CPU section 103
- Understanding bug reports 72
- Understanding call stacks 78

7.1.6 CPU

CPU section shows state of CPU when exception was raised. This section can be disabled in options 266.

Notes:

- CPU section is automatically disabled for reporting leaks;
- Capturing CPU state requires installing of low-level hooks 349.

CPU section contains 3 parts:
1. Registers
2. Stack dump
3. Memory dump

Registers

Registers block contains list of CPU general-purpose registers (EAX-EDX or RAX-R15), including two index registers (ESI-EDI or RSI-RDI) and two stack pointer registers (ESP-EBP or RSP-RBP).

EIP/RIP register shows current instruction pointer. This is exact instruction which raised exception. It will be some instruction inside Kernel32.RaiseException function for software exceptions. It will be any other CPU instruction for hardware exceptions.

FLG value shows content of EFLAGS/RFLAGS service register.

EXP and STK values shows used addresses for exception address and stack. See section below on explanation.

Stack dump

Stack dump block shows dump of memory used by CPU stack. Left column shows addresses, right column shows data stored in stack. Stack is taken from STK position.

Memory dump

Memory dump block shows dump of memory starting from EXP address. Usually this is code section, which is represented in a human-readable form inside Assembler section 100.

Real and logical exception address

Hardware exceptions (such as access violation or divide by zero) ara always raised by failed instruction. Thus, exception address for hardware exception points to failed instruction. For example:

```
var
  P: PInteger;
  E: Extended;
begin
  P := nil;
  P^ := 0;    // <- access violation exception here

  E := 0;
  E := 5 / E; // <- division by zero exception here
end;
```

Both exceptions in the above code will have unique exception address pointing to failed CPU instruction. Thus, you can easily identify failed code for hardware exception.

On the other hand, any software exception is raised from a single place: Kernel32.RaiseException function. This means that all software exceptions will use the same address, which makes it quite useless. Consider the following code:

```
Read(F, Data);
if IOResult <> 0 then
  raise EUnableToReadFile.Create('There was an error reading file');
if Data = 0 then
  raise EInvalidStorageData.Create('File is corrupted, invalid data found');
```

This code raises two software exceptions. Both exceptions will be raised by the same Kernel32.RaiseException function (which is called by Delphi's "raise" keyword). This makes exception address useless because it does not point to failed code.

For this reason: EurekaLog uses logical exception address which is indicated by EXP value above and also listed as exception address in General section 74. Logical exception address is equal to real exception address for all hardware exceptions. However, this address points to appropriate "raise" construction for all software exceptions. This makes it behave similar to hardware exceptions - that is, now exception address always point to failed instruction.

The similar ideas can be applied to stack. STK value indicate stack pointer value for exception which is normally would be ESP/RSP register, but it can be altered for software exceptions to exclude calls of service routines (that is "raise", RaiseException, stack collection routines, etc.).

See also:
- Assembler section 100
- Understanding bug reports 72
- Understanding call stacks 78

7.1.7 Screenshot

ELP bug reports may contain additional files along with usual textual bug report. Additional files may be supplied by your code (for example, to include configuration file or currently opened document) or it can be a screenshot. Screenshot is not different from any other attached files, but it can be displayed by EurekaLog Viewer tool. EurekaLog Viewer consider additional file to be a screenshot if it has "Screenshot.png" or "BugReport.png" file name.

EurekaLog can automatically take screenshot of your application or desktop. This behavior can be set here 304.

Screenshot is sent as separate file in 256-color (8-bit) PNG format (if screenshot creation was enabled, of course). Maximum screenshot file size is typically less than 150 Kb for full screen. Typical file size is around 20 Kb (when saving one average window only). Active control or active window are indicated by bounding red rectangle. Screenshot will contain mouse cursor, if mouse cursor was positioned inside captured screen area. Your application's windows may be covered by other applications. This is especially true for "Capture active window only (may belong to other process)" mode. Screenshot may contain data from multiple monitors (capturing entire desktop or capturing window which is placed across few monitors). Any area outside of any monitors (if it exists) will be filled with black color.

See also:
- Understanding bug reports 72
- Understanding call stacks 78

7.1.8 Additional files

ELP bug reports may contain additional files along with usual textual bug report. Additional files may be supplied by your code (for example, to include configuration file or currently opened document). You can specify static files (such as configuration files) in options 304 or add files at run-time via OnZippedFilesRequest event handlers.

Note:
- A special case of additional files is a screenshot 105;
- You can also use OnAttachedFilesRequest event handler to attach additional files with bug report. This is different from OnZippedFilesRequest event handler which adds file inside bug report.

See also:
- Screenshot 105
- Understanding bug reports 72
- Understanding call stacks 78
- OnZippedFilesRequest
- OnAttachedFilesRequest

7.2 Managing bug reports in issue tracker

Common use case includes the following actions:

1. You need to create "project" in your bug tracker software. One "project" for each of your products or component, which you want to track individually. Optionally you need to assign category and/or other classification to "project" (it depends on your bug tracker software).

2. You need to create "submitter" (reporter) account, which will be used to submit reports. Limit its rights as much as possible. Typically submitter account needs rights to create new bug reports, list and view existing reports (to determinate if issue was already closed) and update/comment existing issue (to change "number of occurrences"/"count" and similar information in existing report). Exact rights depends on your bug tracker software. We recommend to setup account with minimum rights and test send. Increase account's rights in case of insufficient privileges. Test sending again. Repeat until you'll get successful work. See also: Security Considerations 158.

Notes:

- Be sure to test 3 kind of sending:
  - send new report (previously unknown issue);
  - send same report twice (reporting known problem);
  - send report, when issue was closed (to test "bug closed" case).
- You need to do this action only once. However, optionally you can create one submitter account for each "project" (see below). But this is optional.

It is also a good idea to block changes into this account (i.e. disable possibility for the account to alter its password, delete itself, etc.). That is because your EurekaLog-enabled executable will store account details in order to be able to submit reports, which means account details can be stolen and used for malicious actions. See also: Security Considerations 158.

3. Assign "submitter" account to each of your "projects".

Note: it's highly recommended to create standalone category and/or "projects" for automatic submissions. Create another "project" to manage manually created issues. You can move issues between two "projects". This will help to isolate manual and automatic submissions. This is especially useful, if you give many rights to "submitter" account. Separating into two "projects" will ensure that "submitter" account can't mess with your important information. It has access only to "projects" for automatic submissions. However, this is optional action. See also: Security Considerations 158.

4. Create custom informational fields and assign them to your "projects". Exact details depends on your bug tracker software. For example, you can create "count" field for Mantis, BugZilla and JIRA (FogBugz already has such field). Another useful field is e-mail contact. You need to do this action only once.

5. Setup e-mail notifications, if needed.

6. Enter bug tracker details into EurekaLog configuration of your projects.

Common "gotcha's" for using bug tracker software

Here are some points which are worth looking for:

- Try to avoid non-ASCII characters (those which code is above 128) in host/URLs, account names, passwords, etc. While most recent environments offer full support for localized names and characters, older platforms may limit EurekaLog capability to use them. For example, using Cyrillic account name in Delphi 7 will break sending on Italian Windows - because ANSI string will be treated in wrong code page.
- Create a new account specifically for sending reports. **NEVER** use main/personal/administrator account for bug report submission. See also: Security Considerations 158.
- Create a new "project" or account for each of your products. Do not mix several products with one account. For example, create different "projects" in bug-tracker software for each of your software products. See also: Security Considerations 158.
- Test sending before deploying. Test it with both exceptions and leaks. Test it for new and closed reports. Be sure that sending process meets your expectations.
- Before upgrading/changing your end of bug report submission (HTTP upload script, FTP configuration, bug tracker software, etc) - be sure to test this new environment. Ensure that new configuration allows old versions of your application to report bugs (if you still need these bug reports). For this reason be extra careful to use "hosted" solution - because you may not control server software changes.
- There are events for customizing sending: such as OnAttachedFilesRequest,

OnZippedFilesRequest, and OnCustomWebFieldRequest.

See also:
- Using unsupported bug-tracker 153
- Detailed walkthough for setup 107
- Security Considerations 158

Sending algorithm

When you deploy EurekaLog-enabled application to your clients - your application will send you bug reports about each found problem "from the field". Application will try to log in to your bug tracker server, then it search for current bug (to see if this is known issue or not). If issue wasn't found - application creates a new issue and assign it to specified account. If issue was found (reporting known bug) - application will read report to determinate its status (closed or not). If issue isn't closed - application will edit it to update "count" field, upload bug report (optional) and do similar actions (can be set in options - see options for bug tracker sending). If issue is already closed - application will do nothing.

If you've enabled corresponding options - application will display a success/error/"this bug is fixed" message. See also: Customizing Feedback 155.

Working with bug reports

Common bug resolving path 68 is as following: first, application reports about found bug. You'll see new issue/ticket in your bug tracker software (either you visit it regularly or receive e-mail notifications about new problems). Now you can start to study problem, bug report, search for solution. While you're working on the issue, your application will continue to reporting about this problem. Basic reporting includes updating "count" field, but you can also enable gathering additional bug reports, if you like.

Notes:
- Alternatively, when you're in tight time conditions, you can let bug reports to be collected. As the time pass, you'll see which bugs occur most often. So, you can start working on 1-2 "top-bugs". Typically, solving small percentage of your bugs (from your "top" list) solve over 50% of users' problems 68. Thus, you can save your time (by delaying solving rare bugs) and still have great user's satisfaction.
- Report submission for specific issue will continue until this issue is closed.

Eventually, you'll fix bug (or find it to be "unfixable"/"no action required" in some cases). When this happens, you need to release an update to your software and publish it, say, on your web-site. Then you "close" bug in your bug tracker. When your (old) application encounter this bug again and attempt to submit a bug report - it will discover that this problem is already solved and will tell user to get an update, so he'll no longer get this problem.

Basic messages includes common generic phrases, but you can create a custom feedback messages 155.

See also:
- Selecting send method 55
- Configuring sending 53
- Security Considerations 158
- Exception Driven Development 68
- Customizing feedback 155
- Detailed setup manual for supported bug trackers 107

7.2.1 Bug trackers setup

Please refer to a guide specific to your bug tracker tool:
- FogBugz setup 108
- Mantis setup 119
- BugZilla setup 134
- JIRA setup 143

See also:

- Managing bug reports in issue tracker 105
- Selecting send method 55
- Configuring sending 53
- Security Considerations 158
- Exception Driven Development 68
- Customizing feedback 155

7.2.1.1 FogBugz setup

This article is part of Managing bug report in issue tracker 105 series.

See managing bug reports in issue tracker 105 for common information. Please, **read it first**. For common information and setup of FogBugz itself - please see this article 404. The text below assumes that you already completed FogBuz installation.

Below are detailed steps for recommended FogBugz setup for automatic bug report submission. Before going through setup - make sure to upgrade your FogBugz to the latest version.

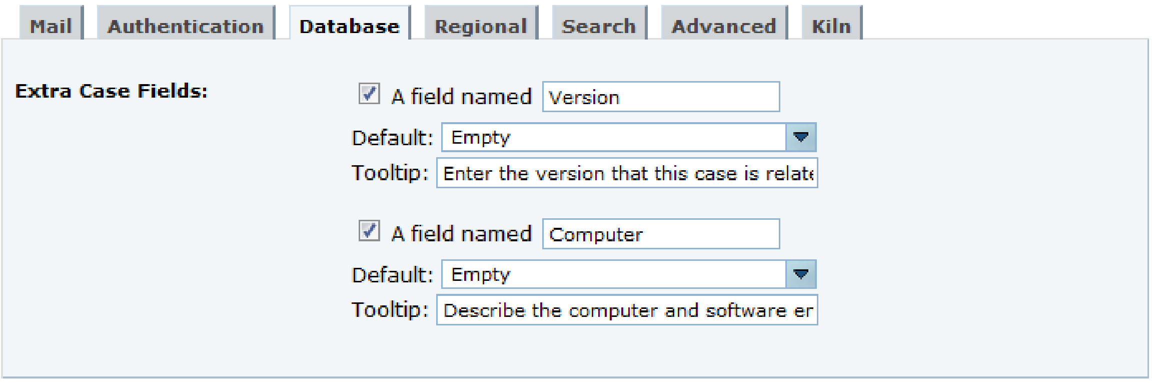
Some steps below are optional, some steps must be executed only once (like custom fields creation), other are executed from time to time (like creating new projects for your new products).

Full list of necessary actions contains:
1. Creating custom fields (single act)
2. Creating user accounts (single act or per product)
3. Creating projects and setting it up (single act or per product)
4. EurekaLog setup (per product)
5. Testing (as required)

Please note that all actions below are just examples. It's recommendation, but it's not necessary to be absolutely like that. You may use another configuration.

Creating custom fields

1. (Admin/Site configuration) Create custom field to improve usefulness of EurekaLog. Most important field is "Version" - to store version of your application (name must be exactly like this, if you want EurekaLog to auto-fill it; otherwise you will need to fill it manually). Other suggested custom field is "Computer" (to store platform information). Again, field name must match or you'll need to manually fill it.



Extra Case Fields:

☑ A field named Version
Default: Empty
Tooltip: Enter the version that this case is relat

☑ A field named Computer
Default: Empty
Tooltip: Describe the computer and software er

Suggested setup for custom fields

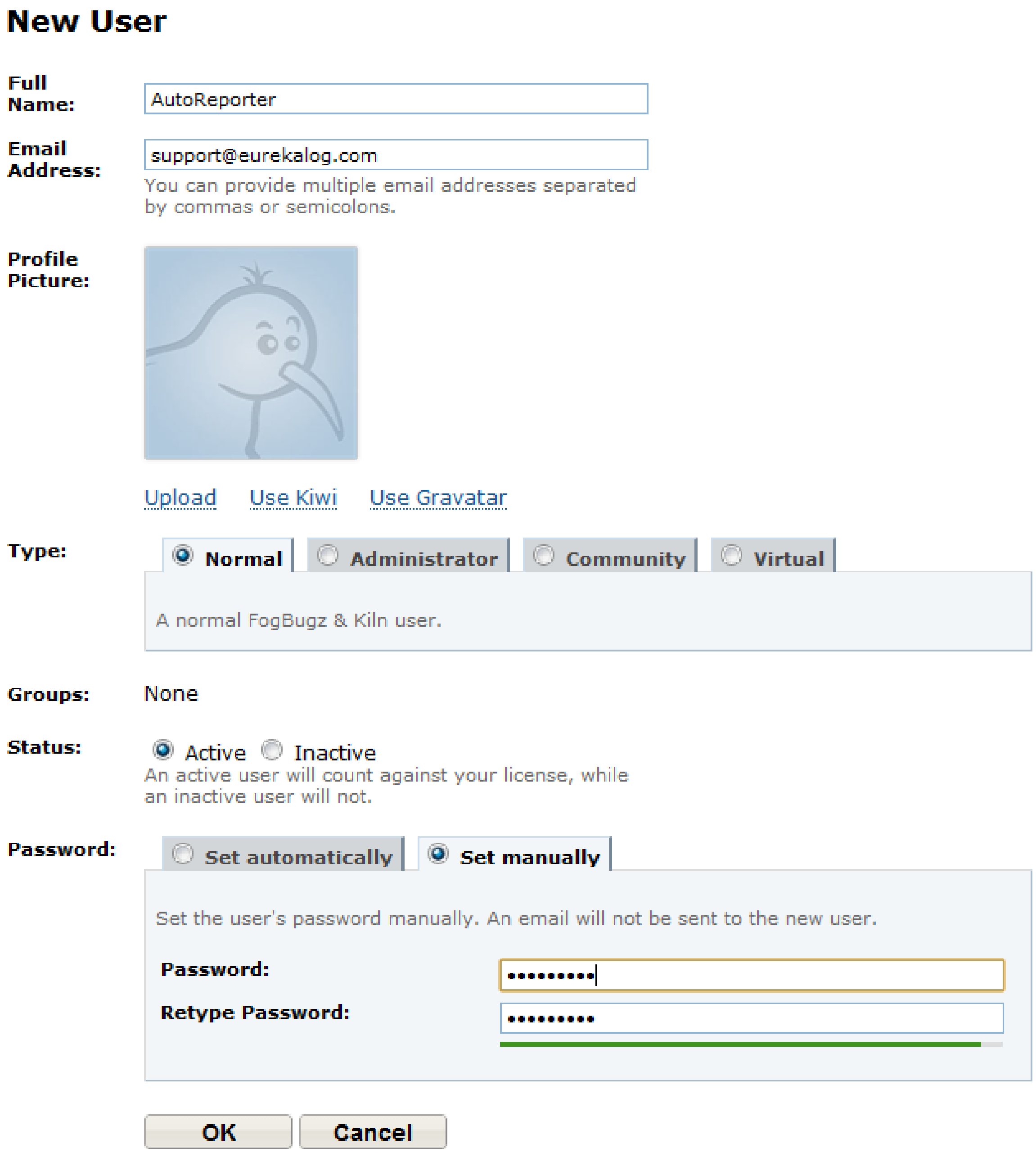
Creating user accounts

1. (Admin/Users) Create new use account for bug report submission. Make it normal or community user. E-mail can be anything. It may be good idea to use dummy e-mail address

(i.e. non-existent e-mail). You may want to create additional accounts for each of your products - for increased security (you can join users in group to simplify control).

Note: you can use anonymous access for bug report submissions (see also). However, currently anonymous user is not supported in FogBugz API (see also). You may also consider using HTTP upload send method 398 to upload report directly to anonymous bug report page 115 for reporting anonymously. Even though limited, anonymous submitting is more secure, since access password isn't stored in your EurekaLog-enabled applications, so end-user can't mess with settings.

Now, back to using real using and full access to FogBugz API.



New User

Full Name: AutoReporter

Email Address: support@eurekalog.com

You can provide multiple email addresses separated by commas or semicolons.

Profile Picture:

Type: ● Normal ○ Administrator ○ Community ○ Virtual

A normal FogBugz & Kiln user.

Groups: None

Status: ● Active ○ Inactive

An active user will count against your license, while an inactive user will not.

Password: ○ Set automatically ● Set manually

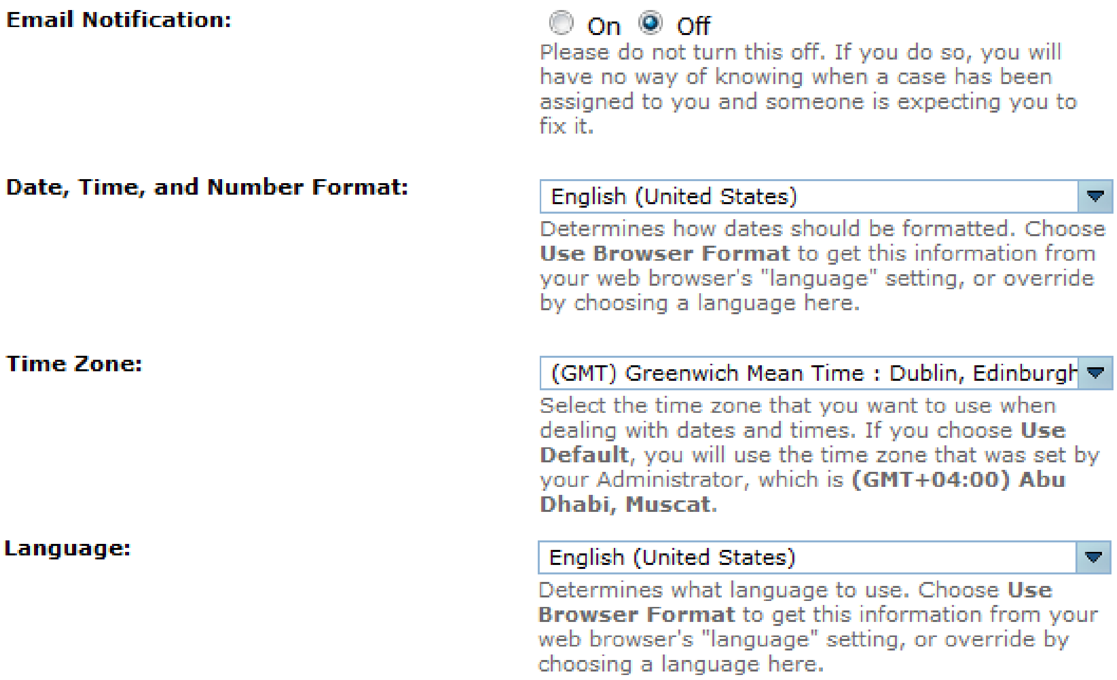
Set the user's password manually. An email will not be sent to the new user.

Password: •••••••••

Retype Password: •••••••••

Creating new user

2. After creating user - click on its name in user list to edit. Turn off e-mail notifications, set settings to defaults, select "English" and "GMT":



Email Notification: ○ On ● Off

Please do not turn this off. If you do so, you will have no way of knowing when a case has been assigned to you and someone is expecting you to fix it.

Date, Time, and Number Format: English (United States)

Determines how dates should be formatted. Choose **Use Browser Format** to get this information from your web browser's "language" setting, or override by choosing a language here.

Time Zone: (GMT) Greenwich Mean Time : Dublin, Edinburgh

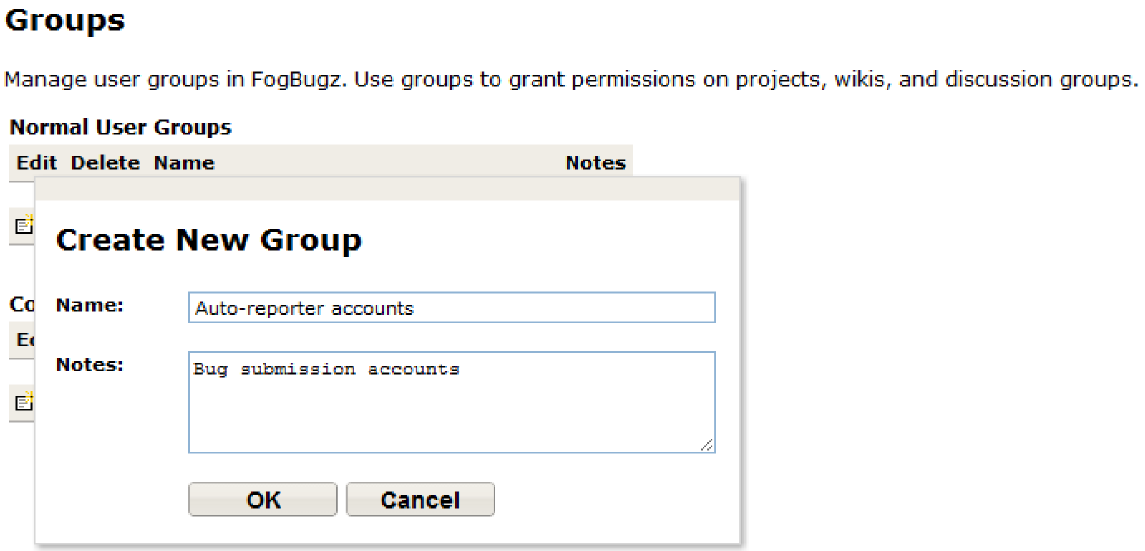
Select the time zone that you want to use when dealing with dates and times. If you choose **Use Default**, you will use the time zone that was set by your Administrator, which is **(GMT+04:00) Abu Dhabi, Muscat.**

Language: English (United States)

Determines what language to use. Choose **Use Browser Format** to get this information from your web browser's "language" setting, or override by choosing a language here.

Suggested settings for auto-reported account

3. You can create new group for all bug submission user accounts (Admin/Groups):



Groups

Manage user groups in FogBugz. Use groups to grant permissions on projects, wikis, and discussion groups.

Normal User Groups

Edit Delete Name Notes

Co

Ec

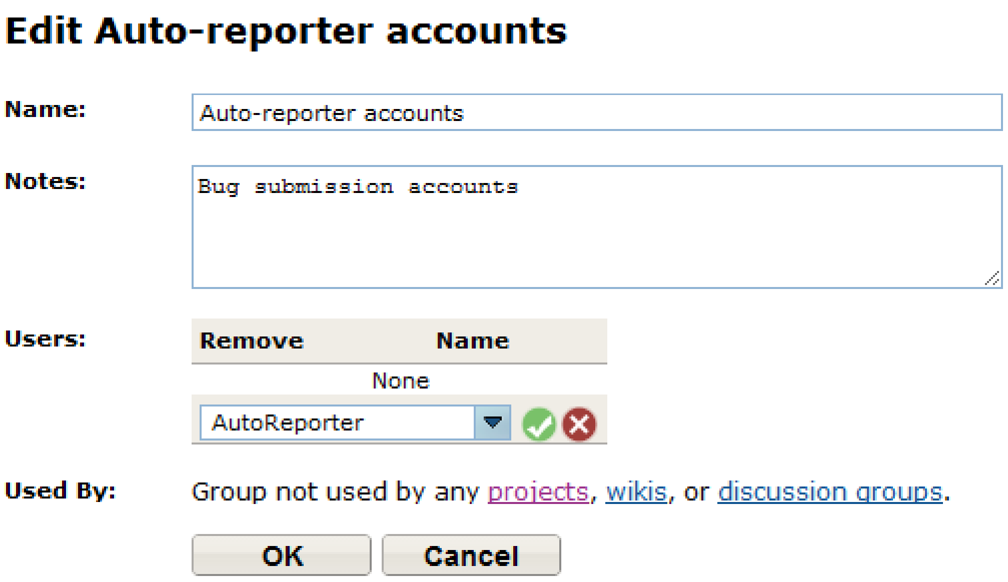
Create New Group

Name: Auto-reporter accounts

Notes: Bug submission accounts

Creating new group

4. After group creation - add users to it:



Edit Auto-reporter accounts

Name: Auto-reporter accounts

Notes: Bug submission accounts

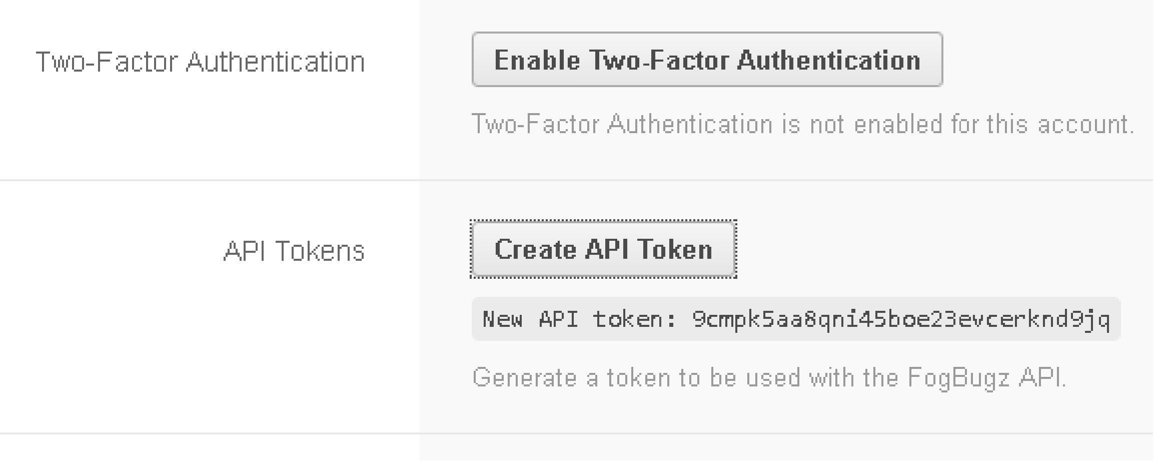
Users:

Remove	Name
	None
AutoReporter	

Used By: Group not used by any projects, wikis, or discussion groups.

Adding user to group

5. (Optional, but strongly recommended; only for latest FogBugz versions) Go to your account create new API token:



Two-Factor Authentication

Two-Factor Authentication is not enabled for this account.

API Tokens

New API token: 9cmpk5aa8qni45boe23evcerknd9jq

Generate a token to be used with the FogBugz API.

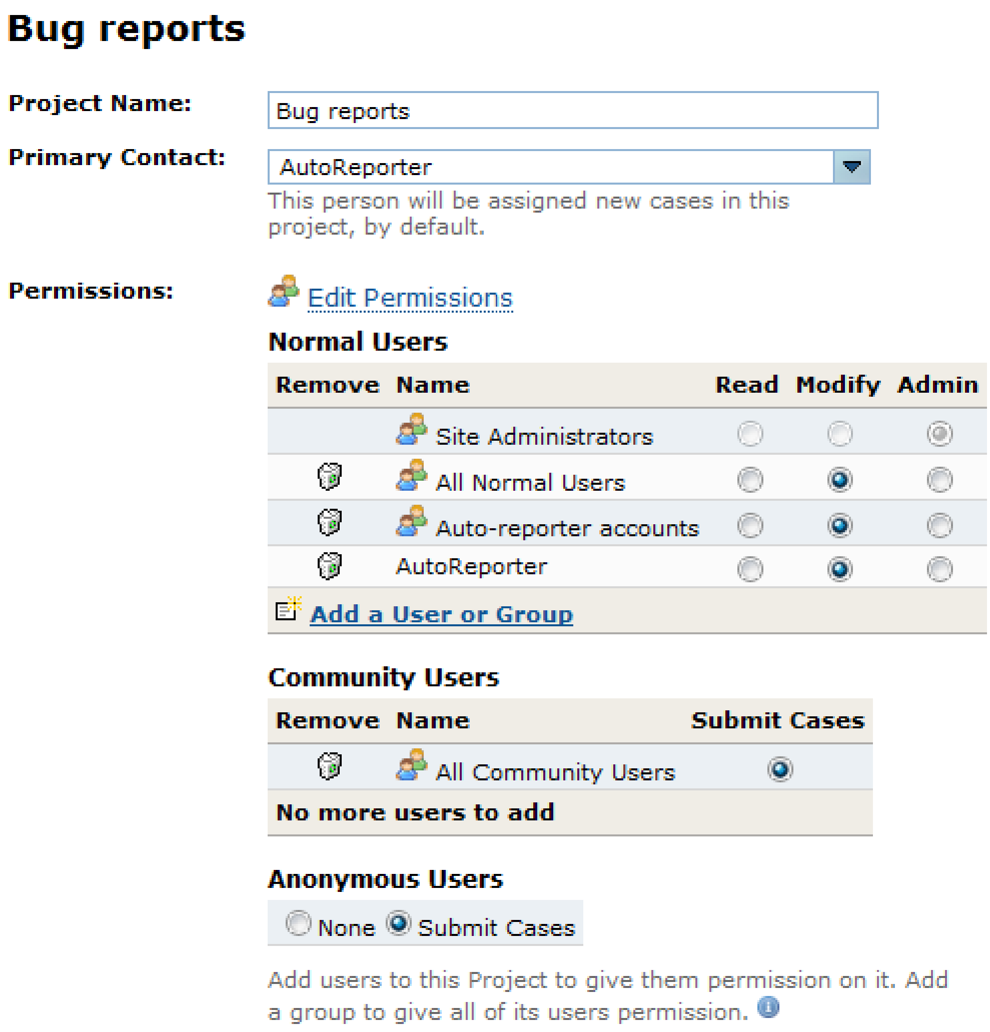
Creating new API token for bug reporter account

Once API token was created - select it and copy to buffer. You will need to enter it into EurekaLog configuration later.

You may create additional tokens.

Creating projects

1. (Admin/Projects) Create project for your product. You may also create several projects - one for each of your products. In this case you may want to create common master project. You can use master project to clone settings to many projects. You can also group projects into a group. Be sure to set appropriate access rights for users (depends on your choice for bug report submission: normal user, community user or anonymous):



Bug reports

Project Name: Bug reports

Primary Contact: AutoReporter

This person will be assigned new cases in this project, by default.

Permissions: Edit Permissions

Normal Users

Remove	Name	Read	Modify	Admin
	Site Administrators	○	○	●
	All Normal Users	○	●	○
	Auto-reporter accounts	○	●	○
	AutoReporter	○	●	○
Add a User or Group				

Community Users

Remove	Name	Submit Cases
	All Community Users	●
No more users to add		

Anonymous Users

○ None ● Submit Cases

Add users to this Project to give them permission on it. Add a group to give all of its users permission.

Example of project for bug report submission

Example above illustrates setting access rights for the following cases:

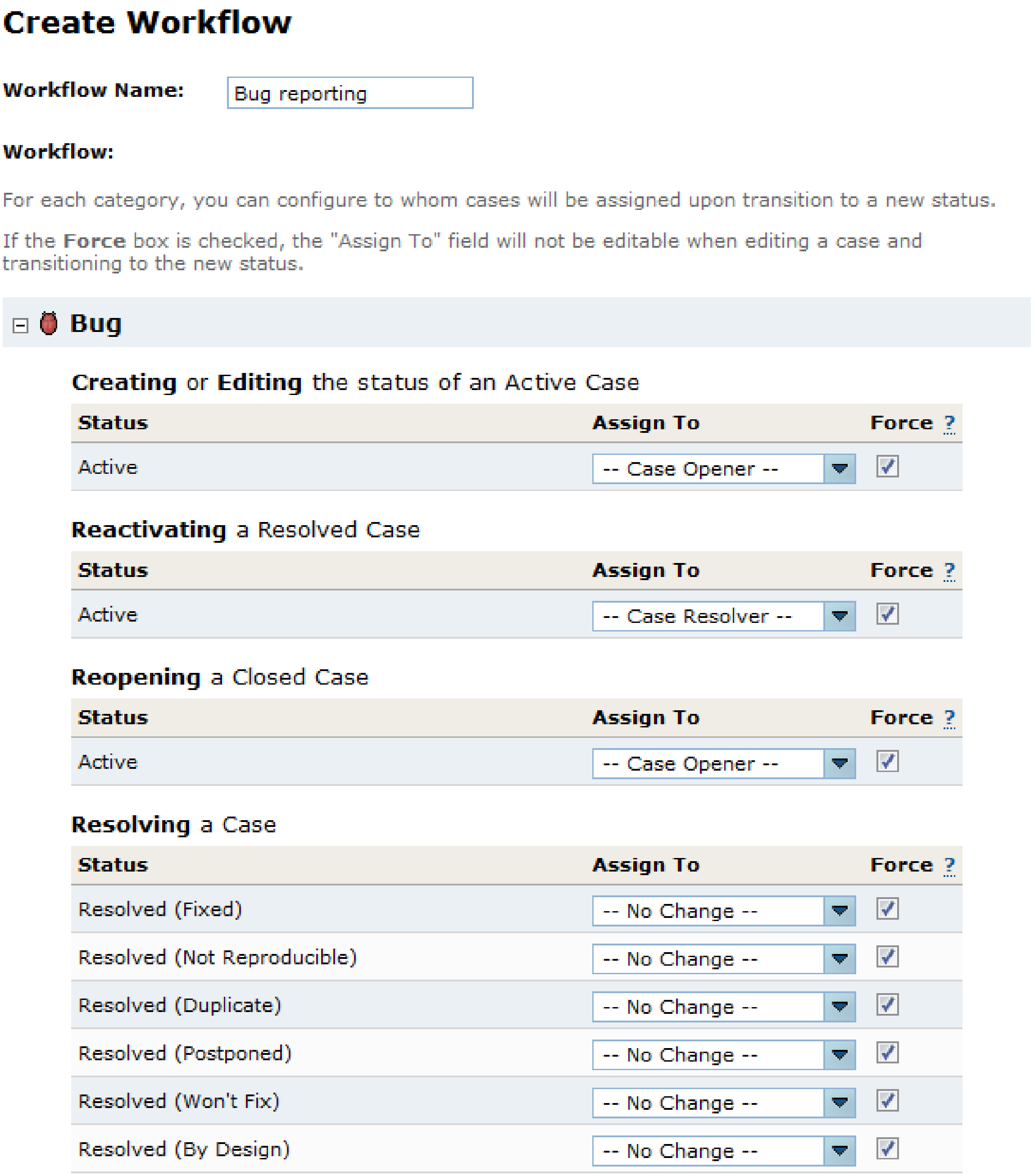
- All normal users
- Special group of user for bug reporting
- Single account
- All community users
- Anonymous user

Please, select only minimum necessary access rights and remove any other.

2. Create area for the projects. If you don't need area - use "Misc" area. If you already have area - assign it. If you have master-project - you can clone project to get pre-set settings.

3. Set any other project properties as needed.

4. (Admin/Workflows) Create custom workflow to limit rights:



Create Workflow

Workflow Name: Bug reporting

Workflow:

For each category, you can configure to whom cases will be assigned upon transition to a new status.

If the **Force** box is checked, the "Assign To" field will not be editable when editing a case and transitioning to the new status.

⊟ **Bug**

Creating or **Editing** the status of an Active Case

Status	Assign To	Force ?
Active	-- Case Opener --	☑

Reactivating a Resolved Case

Status	Assign To	Force ?
Active	-- Case Resolver --	☑

Reopening a Closed Case

Status	Assign To	Force ?
Active	-- Case Opener --	☑

Resolving a Case

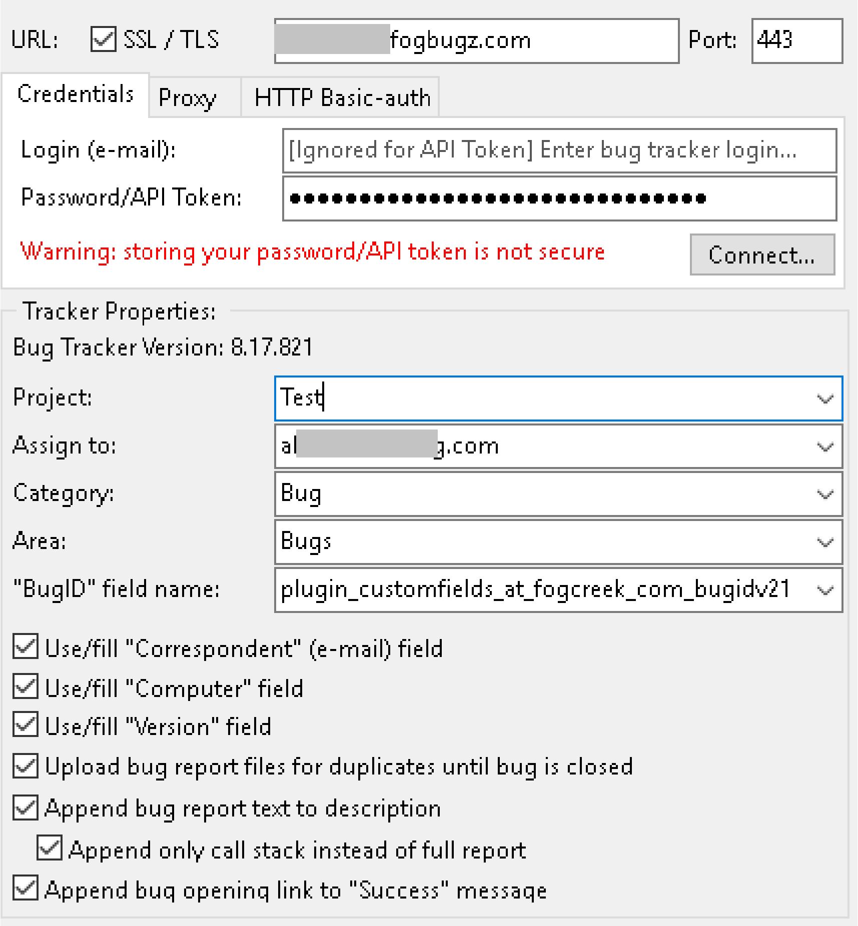
Status	Assign To	Force ?
Resolved (Fixed)	-- No Change --	☑
Resolved (Not Reproducible)	-- No Change --	☑
Resolved (Duplicate)	-- No Change --	☑
Resolved (Postponed)	-- No Change --	☑
Resolved (Won't Fix)	-- No Change --	☑
Resolved (By Design)	-- No Change --	☑

Suggested settings for the workflow for bug reporting projects

Assign workflow to your projects.

EurekaLog setup

1. Enter FogBugz details into EurekaLog configuration of your projects:



URL: ☑ SSL / TLS [redacted]fogbugz.com Port: 443

Login (e-mail): [Ignored for API Token] Enter bug tracker login...

Password/API Token: ••••••••••••••••••••••••••••••

Warning: storing your password/API token is not secure

Tracker Properties:

Bug Tracker Version: 8.17.821

Project: Test

Assign to: al[redacted]g.com

Category: Bug

Area: Bugs

"BugID" field name: plugin_customfields_at_fogcreek_com_bugidv21

☑ Use/fill "Correspondent" (e-mail) field

☑ Use/fill "Computer" field

☑ Use/fill "Version" field

☑ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

  ☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

ForBugz settings filled into EurekaLog options

Important Note: we recommend to use API keys (tokens) when possible. If you are still going to use login/password pair (for example, you are using old FogBugz version, which does not have API keys) - use your user name as login. Even though FogBugz asks for e-mail address as login, we've found out that it's perfectly fine to have several users with same e-mail account, and login with user names instead of e-mail addresses.

2. Set any additional/common send options 304.

3. Set/fill custom fields. EurekaLog has support for automatic managing of "Version", "Computer", "Correspondent" and "BugID" fields. You just need to set corresponding check boxes and field name in EurekaLog options. For other custom fields you need to fill them manually, for example:

```
uses
  EEvents, ESysInfo;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe
begin
  AWebFields.Values['plugin_customfields_at_fogcreek_com_licensev5'] := GetYourAppl
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

4. Add any custom data, additional attached files, write necessary event handlers, set exception filters, etc, etc.

Testing

1. Test sending. You can do this right in the EurekaLog send options dialog 302 - by clicking on "Test..." button. This will send test bug report.

Suggested actions are:

1. Click on "Test..." button to test sending and creating of a new bug issue in FogBugz.
2. Resolve any found issues (access denied, wrong values in fields, etc).
3. Once successful and there is new issue in FogBugz - click on "Test..." button again. This should test updating project.
4. Resolve any found issues (access denied, etc).
5. Once successful - close existing test issue in FogBugz. Optionally - set the BugScout status (see customizing feedback 155).
6. Click on "Test..." button again. This should test sending old (already fixed) bugs.
7. Ensure there is no error messages, no problems. You should get "success, this bug is fixed" kind of behaviour. Exact behaviour depends on your settings.
8. Delete test issue in FogBugz after testing.

These actions should test that sending is actually working.

2. Now it's time to test your application-specific sending.

1. Place debug code in your application to raise test exception and cause a test leak (if you've enabled leaks collecting).
2. Run your application and invoke this test code.
3. Let application crash and process bug (show dialog, send bug report, etc).
4. Ensure that behaviour is expected.
5. Ensure that you get all files and additional information in FogBugz.
6. Remove test code from your application.

Now your application is ready for deployment.

See also:

- Managing bug reports in issue tracker 105
- Security Considerations 158
- Customizing feedback 155

7.2.1.1.1 Using HTTP upload

This article is part of Managing bug report in issue tracker 105 series.

FogBugs 404 allows you to submit reports anonymously without using API. This is an alternative method to submit bug reports.

Note: anonymous submitting may be disabled - please, refer to your FogBugz configuration and documentation.

However, FogBugz doesn't allow you to login anonymously via FogBugz API 108. So, the only choice for anonymous reporting is to submit report directly to web form (web-form for anonymous users is available at this URL: https://your-account.fogbugz.com/default.asp?pg=pgPublicEdit - with user logged off).

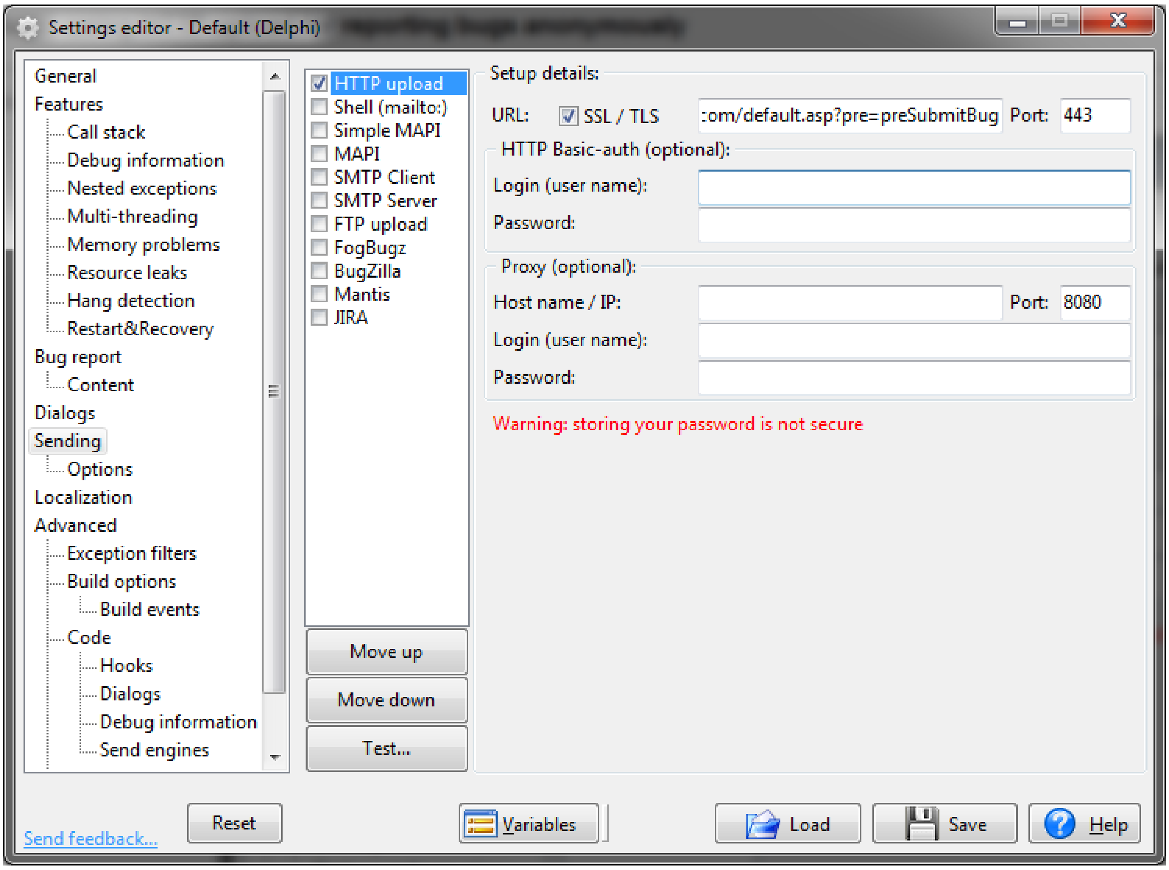
Warning: each sent report will be tracked individually - as new issue in FogBugz. Occurrences field will be 1 always, for all such reports.

Alternatively, you can use BugzScout submission. This way also allows anonymous access, but it also automates merging similar reports. However, file attaches are not supported.

Below are examples for each method.

Web-form

First, start with specifying HTTP upload method:



☑ HTTP upload
□ Shell (mailto:)
□ Simple MAPI
□ MAPI
□ SMTP Client
□ SMTP Server
□ FTP upload
□ FogBugz
□ BugZilla
□ Mantis
□ JIRA

Setup details:

URL: ☑ SSL / TLS :om/default.asp?pre=preSubmitBug Port: 443

HTTP Basic-auth (optional):

Login (user name):

Password:

Proxy (optional):

Host name / IP: Port: 8080

Login (user name):

Password:

Warning: storing your password is not secure

Typical HTTP upload setup for submitting report anonymously in FogBugz

URL is "your-account.fogbugz.com/default.asp?pre=preSubmitBug" (without quotes).

Second, you need to supply custom fields (it's only an example - use your own values for fields):

```
uses
  EEvents, ETypes, EClasses, ESend, ESendWebHTTP;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe

  function ComposeTitle(const AOptions: TEurekaModuleOptions): String;
  var
    BugAppVersion: String;
    BugType: String;
    BugID: String;
  begin
    BugAppVersion := AOptions.CustomField[sifBugAppVersion];
    BugType := AOptions.CustomField[sifBugType];
    BugID := AOptions.CustomField[sifBugID];

    if BugAppVersion <> '' then
      Result := Format('%s (Bug %s; v%s)', [BugType, BugID, BugAppVersion])
    else
      Result := Format('%s (Bug %s)', [BugType, BugID]);
  end;

  function ComposeMessage(const AOptions: TEurekaModuleOptions): String;
  var
    TrackerAppendText: Boolean;
    BugText: String;
    ReproduceText: String;
    Message: String;
  begin
    TrackerAppendText := AOptions.SendFogBugzAppendText;
    BugText := AOptions.CustomField[sifBugText];
    ReproduceText := AOptions.CustomField[sifStepsToReproduce];
    Message := AOptions.CustomField[sifMessage];

    if TrackerAppendText then
      Result := BugText + sLineBreak + sLineBreak + Message
    else
    if ReproduceText <> '' then
      Result := BugText + sLineBreak + sLineBreak + ReproduceText
    else
      Result := BugText;
  end;

var
  Options: TEurekaModuleOptions;
  AttachedFiles: TStringList;
begin
  Options := nil;
  AttachedFiles := TStringList.Create;
  try
    if ASender is TELUniversalSender then
    begin
      Options := TELUniversalSender(ASender).Options;
      AttachedFiles.Assign(TELUniversalSender(ASender).AttachedFiles);
    end
    else
      Options := TEurekaModuleOptions.Create('');

    // hidden fields:
    AWebFields.Values['command'] := 'new';
    AWebFields.Values['honeyInput'] := '';
    AWebFields.Values['honeyTextArea'] := '';
    AWebFields.Values['ixScreenshot'] := '-1';
    AWebFields.Values['dblTimeStamp'] := FloatToStr(Now);
    AWebFields.Values['fPublic'] := '1';
    AWebFields.Values['OK'] := 'OK';

    // fixed fields:
    AWebFields.Values['sTitle'] := ComposeTitle(Options);
    AWebFields.Values['sEvent'] := ComposeMessage(Options);
    AWebFields.Values['sCustomerEmail'] := Options.CustomField[sifUserEMail];
    AWebFields.Values['sVersion'] := Options.CustomField[sifBugAppVersion];
    AWebFields.Values['sComputer'] := Options.CustomField[sifMachineID];
    AWebFields.Values['nFileCount'] := IntToStr(AttachedFiles.Count);
    Options.CustomField[sifHTTPFileNameTemplate] := 'File%d';

    // editable fields - refer to page source (https://your-account.fogbugz.com/def
    AWebFields.Values['ixProject'] := '3';
    AWebFields.Values['ixArea'] := '8';
  finally
    FreeAndNil(AttachedFiles);
    if not (ASender is TELUniversalSender) then
      FreeAndNil(Options);
  end;
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

Done!

Note: study submission form's page source to get information on field names and values. Form is available at https://your-account.fogbugz.com/default.asp?pg=pgPublicEdit with user logged off.

BugzScout

First, start with specifying HTTP upload method. It's the same as above, only this time URL is "your-account.fogbugz.com/scoutSubmit.asp" (without quotes).

Second, you need to supply custom fields (it's only an example - use your own values for fields):

```
uses
  EEvents, ETypes, EClasses, ESend, ESendWebHTTP;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe

  function ComposeTitle(const AOptions: TEurekaModuleOptions): String;
  var
    BugAppVersion: String;
    BugType: String;
    BugID: String;
  begin
    BugAppVersion := AOptions.CustomField[sifBugAppVersion];
    BugType := AOptions.CustomField[sifBugType];
    BugID := AOptions.CustomField[sifBugID];

    if BugAppVersion <> '' then
      Result := Format('%s (Bug %s; v%s)', [BugType, BugID, BugAppVersion])
    else
      Result := Format('%s (Bug %s)', [BugType, BugID]);
  end;

  function ComposeMessage(const AOptions: TEurekaModuleOptions): String;
  var
    TrackerAppendText: Boolean;
    BugText: String;
    ReproduceText: String;
    Message: String;
  begin
    TrackerAppendText := AOptions.SendFogBugzAppendText;
    BugText := AOptions.CustomField[sifBugText];
    ReproduceText := AOptions.CustomField[sifStepsToReproduce];
    Message := AOptions.CustomField[sifMessage];

    if TrackerAppendText then
```

```
      Result := BugText + sLineBreak + sLineBreak + Message
    else
    if ReproduceText <> '' then
      Result := BugText + sLineBreak + sLineBreak + ReproduceText
    else
      Result := BugText;
  end;

var
  Options: TEurekaModuleOptions;
  AttachedFiles: TStringList;
begin
  Options := nil;
  AttachedFiles := TStringList.Create;
  try
    if ASender is TELUniversalSender then
    begin
      Options := TELUniversalSender(ASender).Options;
      AttachedFiles.Assign(TELUniversalSender(ASender).AttachedFiles);
    end
    else
      Options := TEurekaModuleOptions.Create('');

    AWebFields.Values['ScoutUserName'] := 'AutoReporter';
    AWebFields.Values['ScoutProject'] := 'Bug reports';
    AWebFields.Values['ScoutArea'] := 'Misc';
    AWebFields.Values['Description'] := ComposeTitle(Options);
    AWebFields.Values['Extra'] := ComposeMessage(Options);
    AWebFields.Values['Email'] := Options.CustomField[sifUserEMail];
    AWebFields.Values['ForceNewBug'] := '0';
    AWebFields.Values['ScoutDefaultMessage'] := 'html Default Message';
    AWebFields.Values['FriendlyResponse'] := '1';
  finally
    FreeAndNil(AttachedFiles);
    if not (ASender is TELUniversalSender) then
      FreeAndNil(Options);
  end;
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

Done!

Note: see BugzScout documentation and BugzScout sample to get information on field names.

See also:

- FogBugz description 404
- FogBugz setup 108

7.2.1.2 Mantis setup

This article is part of Managing bug report in issue tracker 105 series.

See managing bug reports in issue tracker 105 for common information. Please, **read it first**. For common information and setup of Mantis itself - please see this article 406. The text below assumes that you already completed Mantis installation.

Below are detailed steps for recommended Mantis setup for automatic bug report submission. Before going through setup - make sure to upgrade your Mantis to the latest version (or at least 1.2.7) - there are some important bugs fixed.

Some steps below are optional, some steps must be executed only once (like custom fields creation), other are executed from time to time (like creating new projects for your new products) and the rest are executed regularly (like creating product versions).

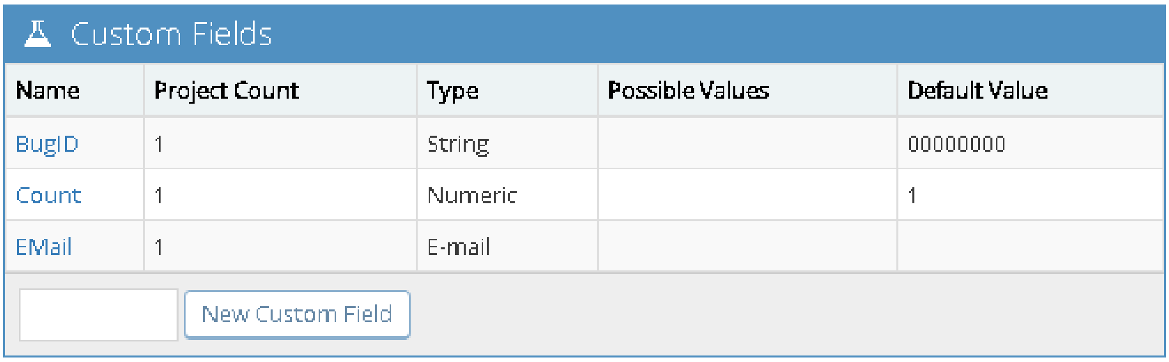
Full list of necessary actions contains:
1. Creating custom fields (single act)
2. Creating user accounts (single act or per product)
3. Creating projects and setting it up (single act or per product)
4. EurekaLog setup (per product)
5. Testing (as required)
6. Maintaining project (regularly or from time to time)

Please note that all actions below are just examples. It's recommendation, but it's not necessary to be absolutely like that. You may use another configuration.

Creating custom fields

1. (Manage/Manage Custom Fields) Create custom field to improve usefulness of EurekaLog. Most important field is "Count" - to store number of sent/occurred problems. Its type should be "Numeric" (integer); field name can be arbitrary, like "Occurrences", "Bug count", "Popularity", "Incidents", "Hit Count", etc. Other suggested custom fields are: "BugID" (to store BugID and search issues) and "e-mail" ("user e-mail", which is typically entered in MS Classic error dialog 377). "BugID" field should be text field ("String"), "e-mail" field should be either "String" type or "E-mail" type. Again, field names can be anything.

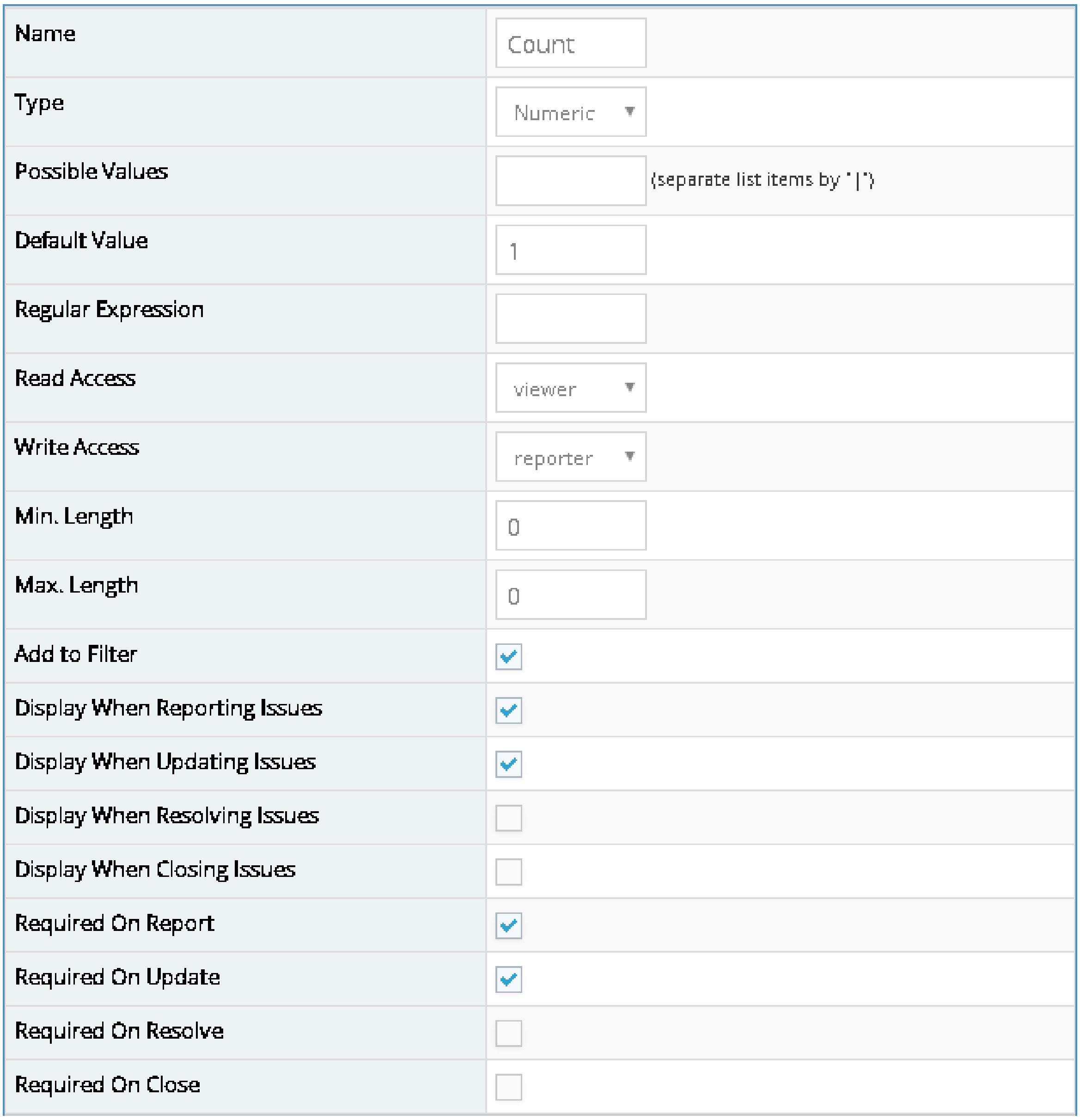
We strongly recommend to create at least "Count" and "BugID" fields.



Custom Fields

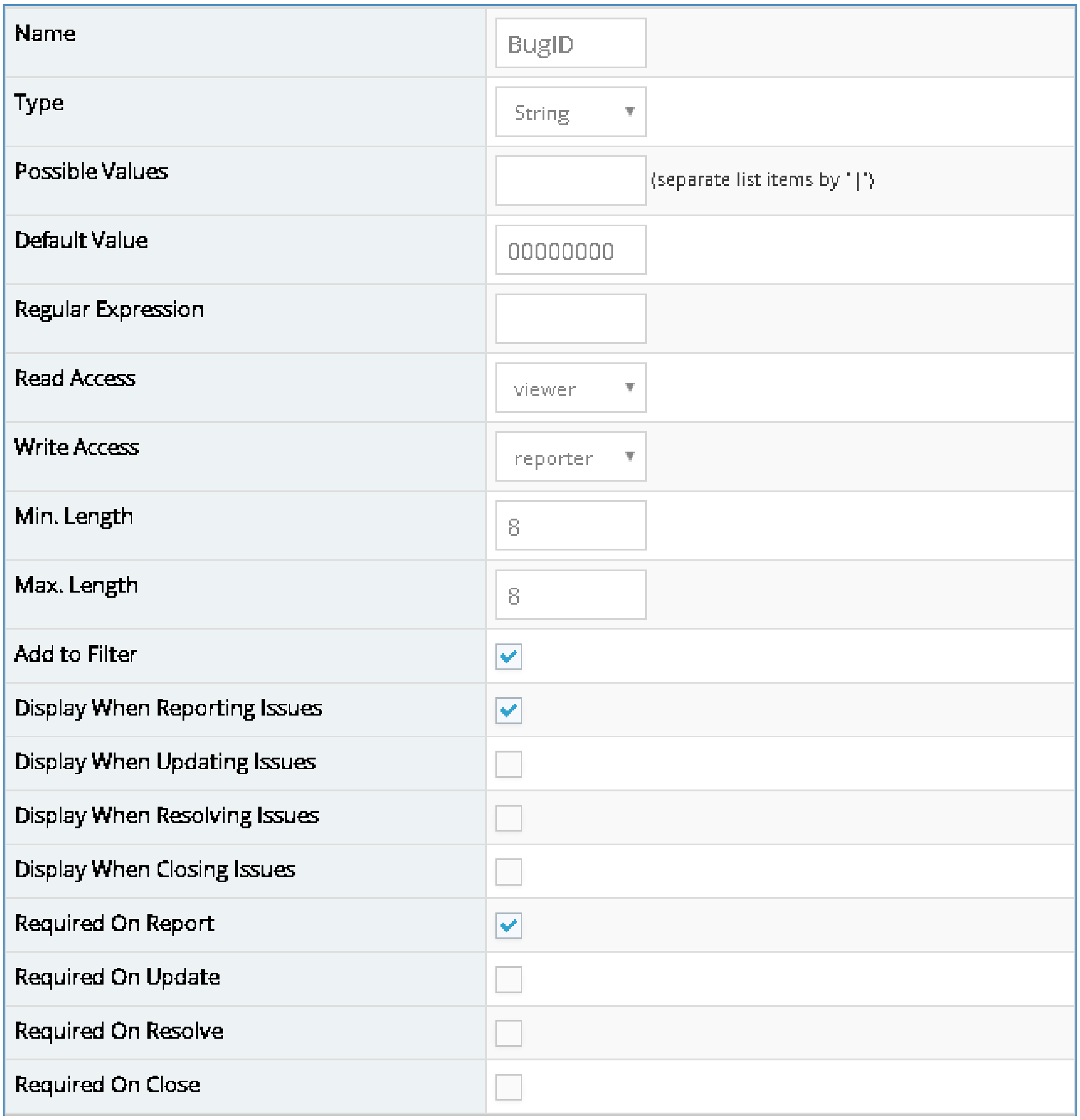
Name	Project Count	Type	Possible Values	Default Value
BugID	1	String		00000000
Count	1	Numeric		1
EMail	1	E-mail		

Example: three new custom fields



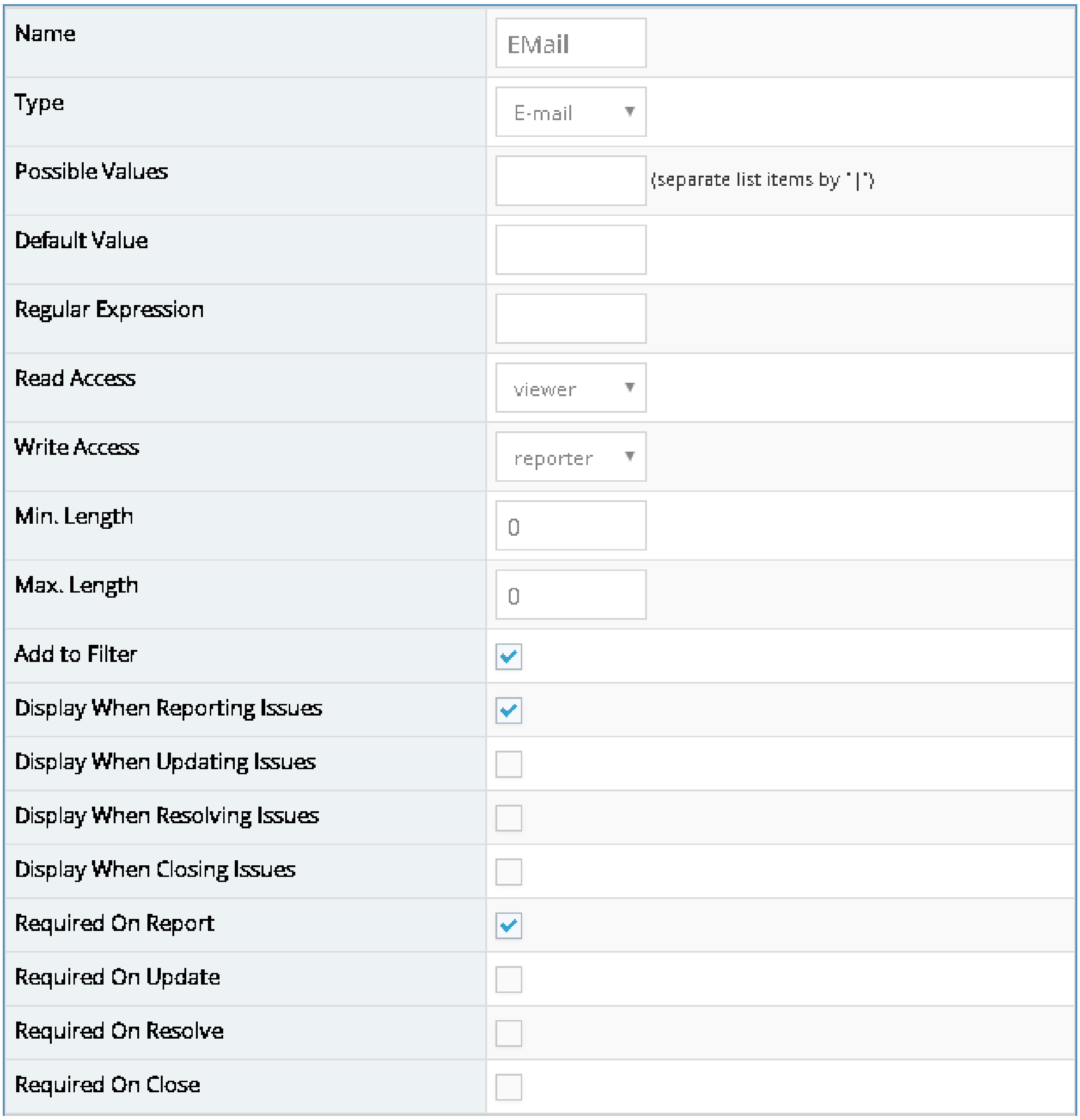
Name	Count
Type	Numeric ▼
Possible Values	(separate list items by " | ")
Default Value	1
Regular Expression	
Read Access	viewer ▼
Write Access	reporter ▼
Min. Length	0
Max. Length	0
Add to Filter	☑
Display When Reporting Issues	☑
Display When Updating Issues	☑
Display When Resolving Issues	□
Display When Closing Issues	□
Required On Report	☑
Required On Update	☑
Required On Resolve	□
Required On Close	□

Suggested setup for "Count" field



Name	BugID
Type	String ▾
Possible Values	(separate list items by "|")
Default Value	00000000
Regular Expression	
Read Access	viewer ▾
Write Access	reporter ▾
Min. Length	8
Max. Length	8
Add to Filter	☑
Display When Reporting Issues	☑
Display When Updating Issues	□
Display When Resolving Issues	□
Display When Closing Issues	□
Required On Report	☑
Required On Update	□
Required On Resolve	□
Required On Close	□

Suggested setup for "BugID" field



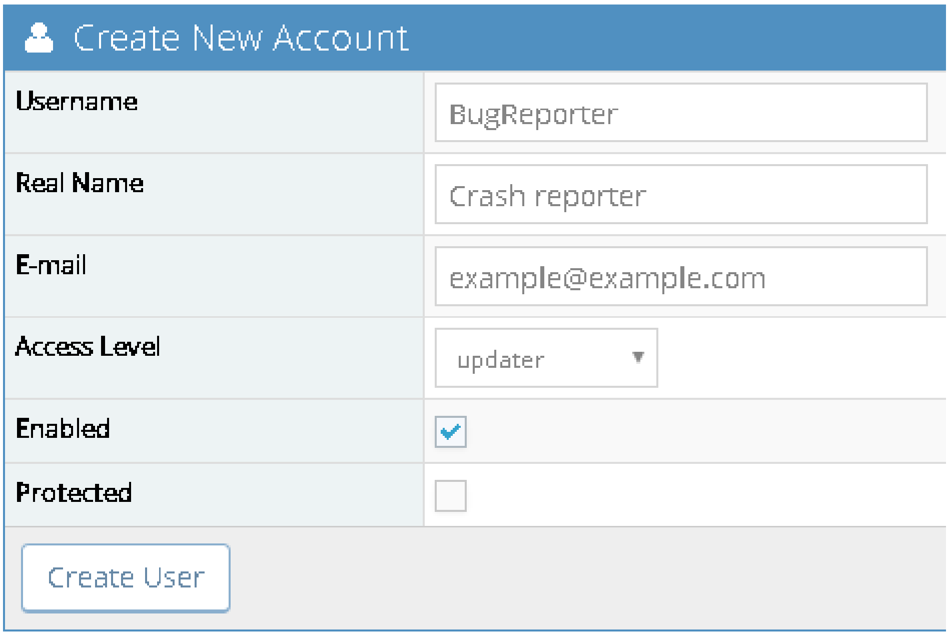
Name	EMail
Type	E-mail ▾
Possible Values	(separate list items by "|")
Default Value	
Regular Expression	
Read Access	viewer ▾
Write Access	reporter ▾
Min. Length	0
Max. Length	0
Add to Filter	☑
Display When Reporting Issues	☑
Display When Updating Issues	□
Display When Resolving Issues	□
Display When Closing Issues	□
Required On Report	☑
Required On Update	□
Required On Resolve	□
Required On Close	□

Suggested setup for "e-mail" field

2. Create any other additional fields as you need/like (you can submit values for custom fields at run-time via OnCustomWebFieldsRequest event).

Creating user accounts

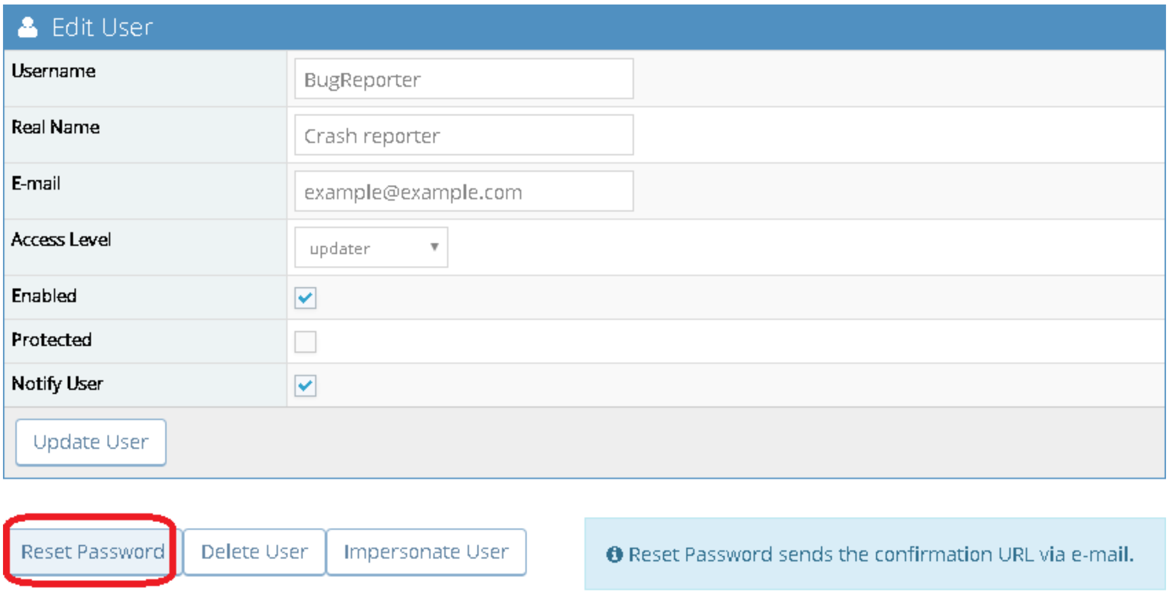
1. (Manage/Manage Users) Create new use account for bug report submission. Make it "updater" or "reporter" level. "Reporter" level can only report about new bugs (create new bug reports). "Updater" level can alter existing bug reports - for example, updating "Count" field (see above). You may want to create additional accounts for each of your products - for increased security. Use your e-mail to setup initial password. Once password is set, you can set e-mail to any value.



Create New Account

Username	BugReporter
Real Name	Crash reporter
E-mail	example@example.com
Access Level	updater
Enabled	☑
Protected	□

Creating new user



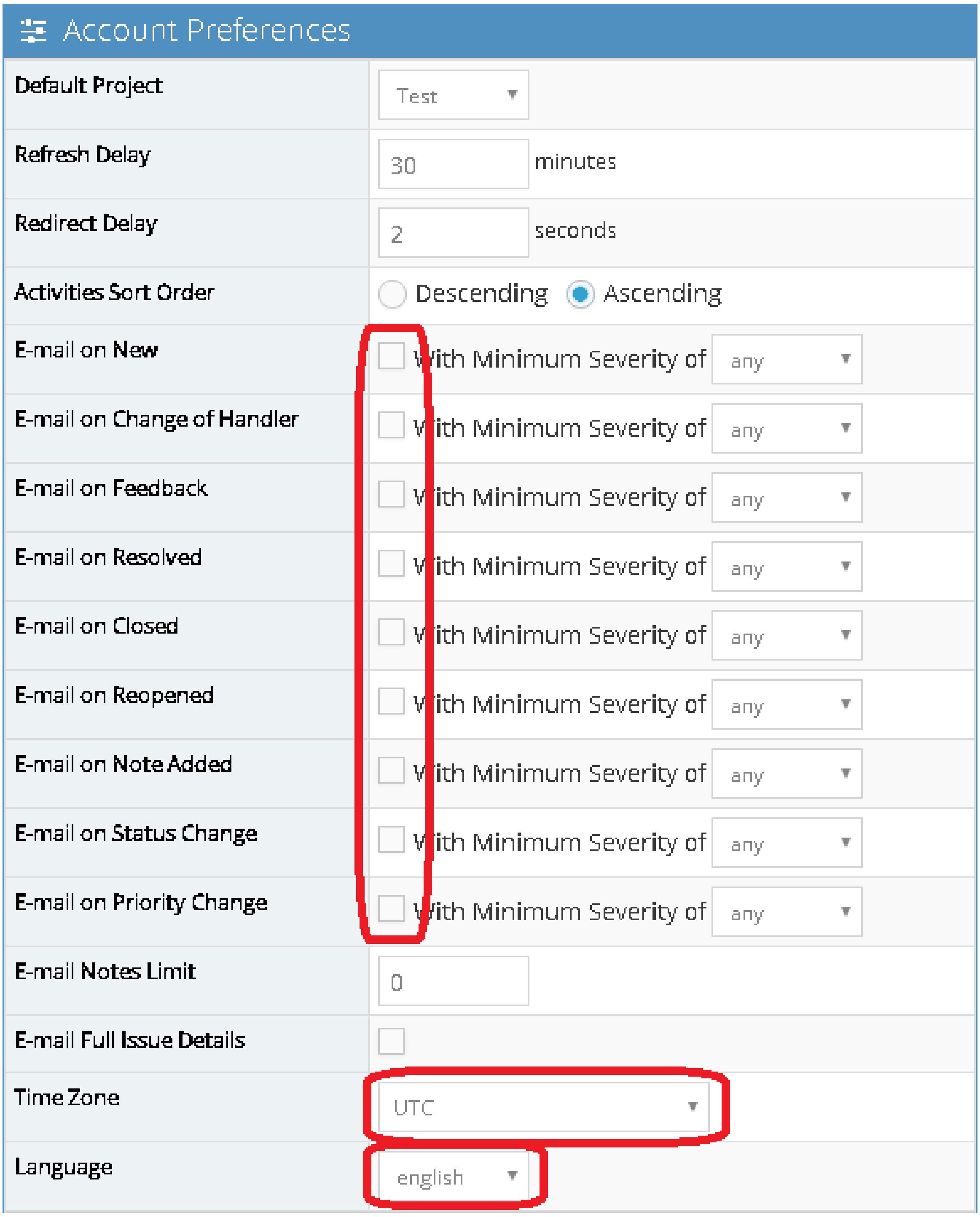
Edit User

Username	BugReporter
Real Name	Crash reporter
E-mail	example@example.com
Access Level	updater
Enabled	☑
Protected	□
Notify User	☑

Reset Password sends the confirmation URL via e-mail.

Setting new password for the user

2. Once password is set - log off and log in as this user. Go to "My account" / "Preferences" and clear e-mail notifications, set default project, switch language to "English", and time zone to "UTC":

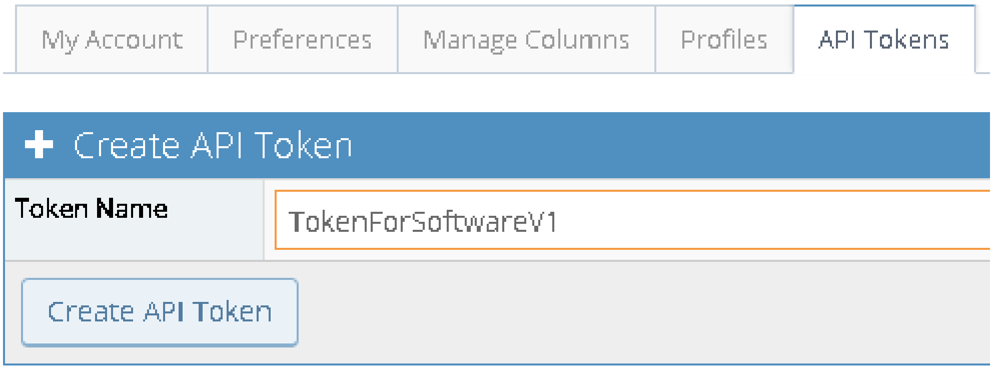


Account Preferences

Default Project	Test
Refresh Delay	30 minutes
Redirect Delay	2 seconds
Activities Sort Order	○ Descending ● Ascending
E-mail on New	□ With Minimum Severity of any
E-mail on Change of Handler	□ With Minimum Severity of any
E-mail on Feedback	□ With Minimum Severity of any
E-mail on Resolved	□ With Minimum Severity of any
E-mail on Closed	□ With Minimum Severity of any
E-mail on Reopened	□ With Minimum Severity of any
E-mail on Note Added	□ With Minimum Severity of any
E-mail on Status Change	□ With Minimum Severity of any
E-mail on Priority Change	□ With Minimum Severity of any
E-mail Notes Limit	0
E-mail Full Issue Details	□
Time Zone	UTC
Language	english

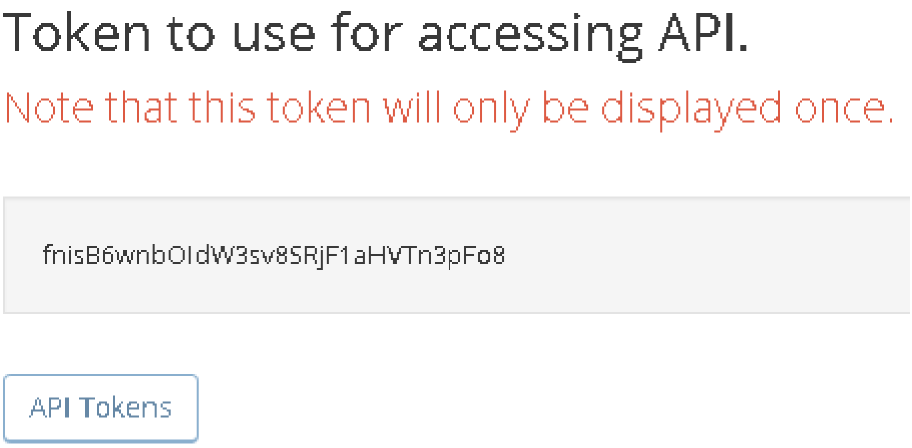
Setting properties of submitter account

3. (Optional, but strongly recommended; only for latest Mantis versions) Go to "My account" / "API Tokens" and create new API token:



Token Name: TokenForSoftwareV1

Creating new API token for bug reporter account



Token to use for accessing API.

Note that this token will only be displayed once.

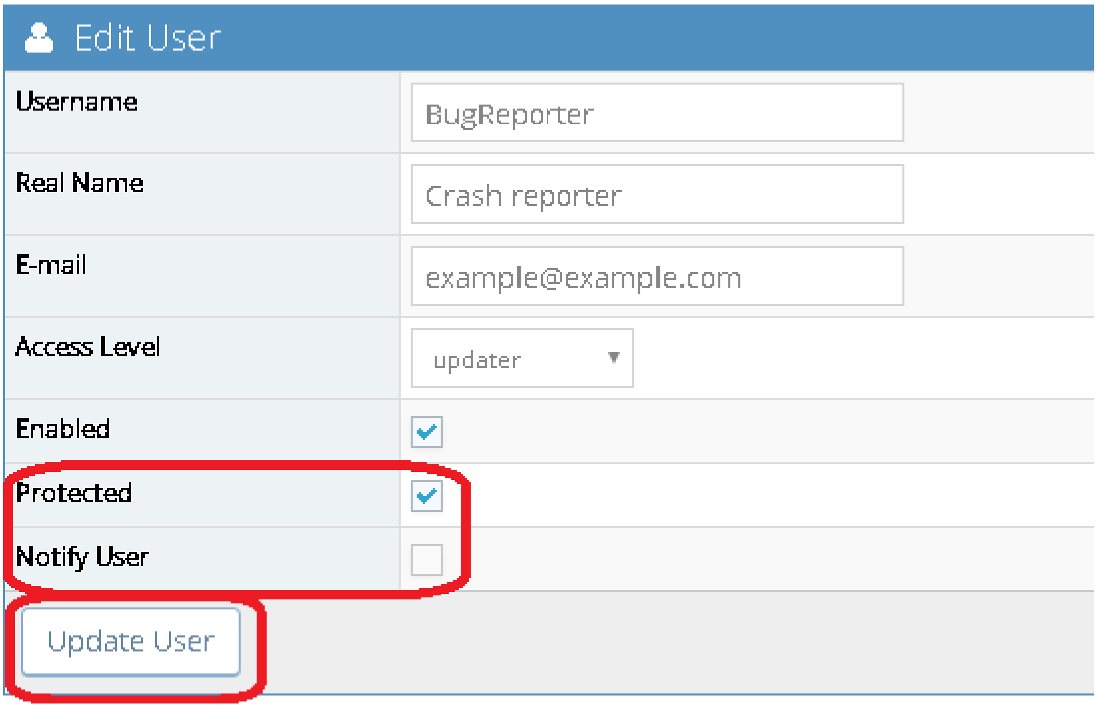
fnisB6wnbOIdW3sv8SRjF1aHVTn3pFo8

New API token was created

Once API token was created - select it and copy to buffer. You will need to enter it into EurekaLog configuration later.

You may create additional tokens.

4. Now, log off and log in again as administrator. Go to managing user and turn on "protected" checkbox for this user account (which means that user will not be able to change password); turn off "Notify user".



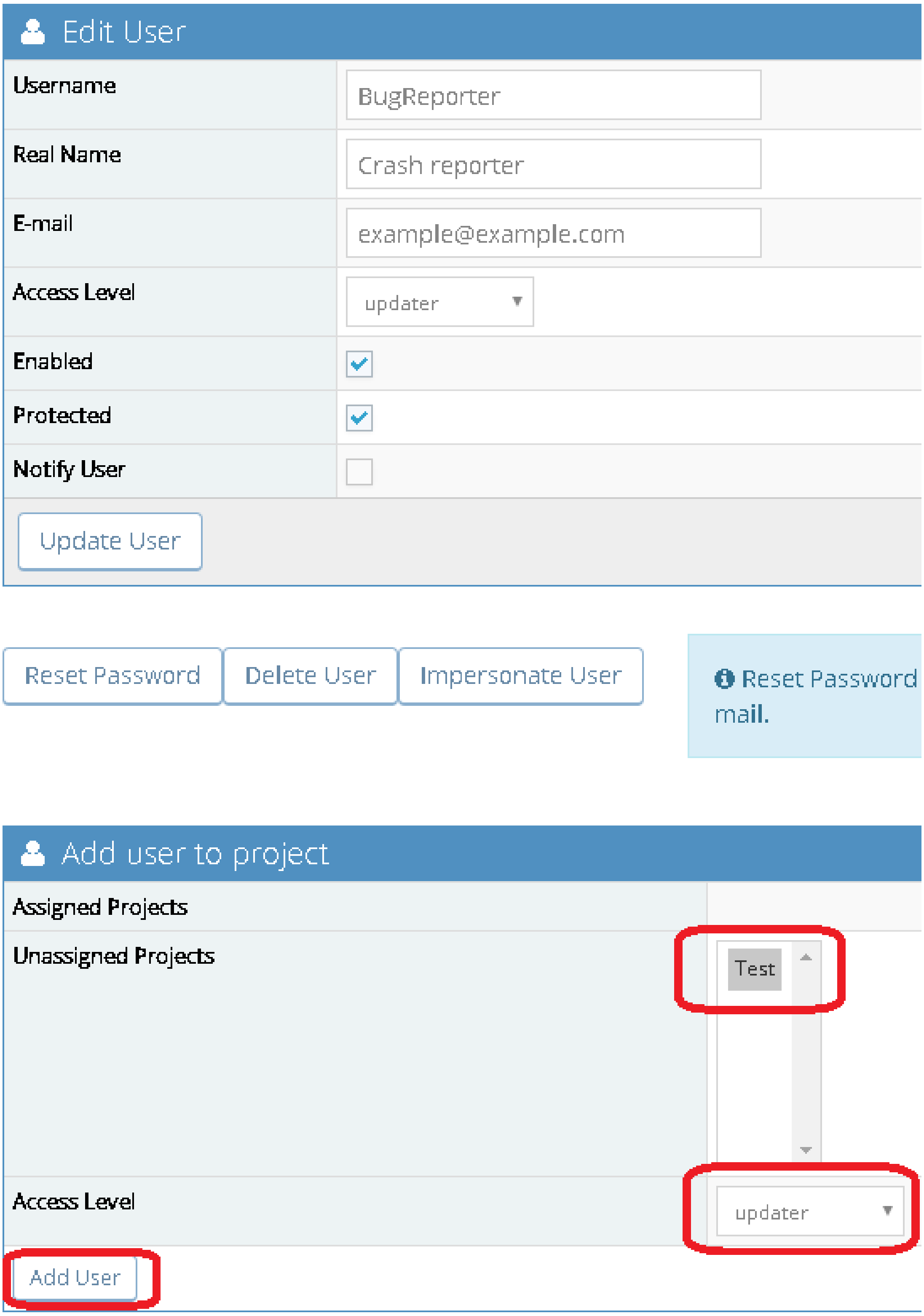
Edit User

Username	BugReporter
Real Name	Crash reporter
E-mail	example@example.com
Access Level	updater
Enabled	☑
Protected	☑
Notify User	□

Make bug reporting account "protected"

Additionally, you may change e-mail field to dummy e-mail address.

[Optionally] And assign user to project (you may need to make this step after project was created - see below).



Edit User

Username	BugReporter
Real Name	Crash reporter
E-mail	example@example.com
Access Level	updater
Enabled	☑
Protected	☑
Notify User	□

Add user to project

Assigned Projects	
Unassigned Projects	Test
Access Level	updater

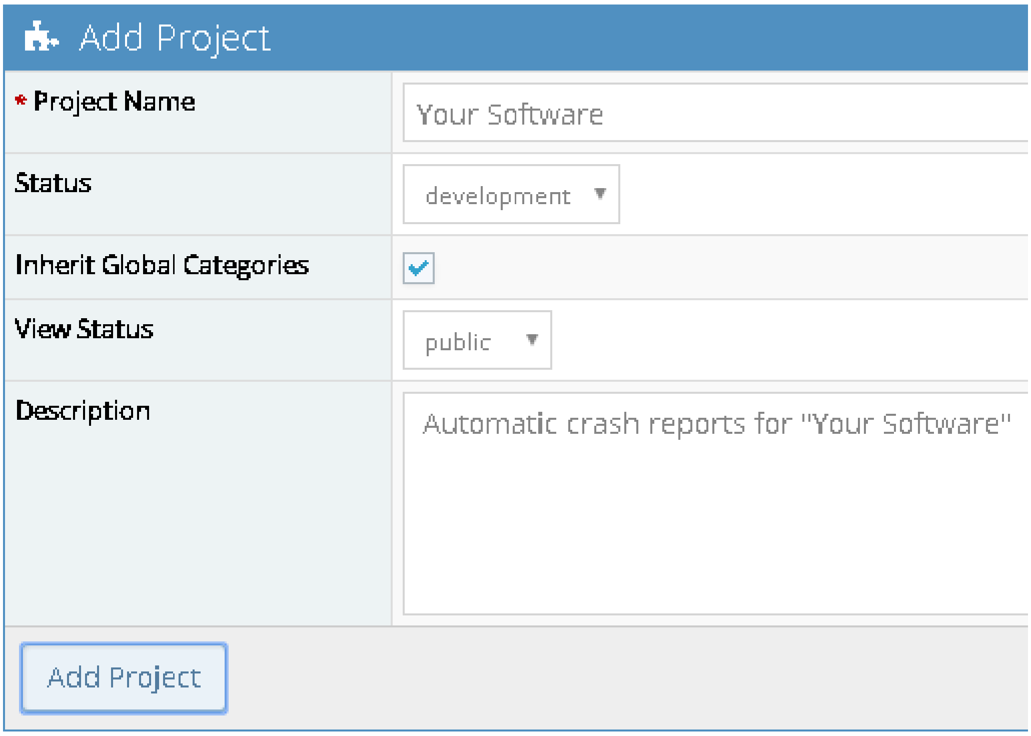
Assigning user to project

Repeat these steps for each bug submitter user account which you've created.

Note: Mantis also support anonymous logins. See also. Anonymous user is predefined user with "anonymous" or "guest" name and without the password. However, since Mantis supports protected users (which can't alters their account settings), anonymous posting is not recommended.

Creating projects

1. (Manage/Manage Projects) Create project for your product. You may create it as private (thus, it won't be visible to users except those who are explicitly assigned for it). You may also create several projects - one for each of your products. In this case you may want to create them as sub-projects of common master project. You can use master project to setup versions/category/etc and copy these settings to sub-projects. Be sure to assign reporter user account as "reporter" or "updater".



* Project Name	Your Software
Status	development ▼
Inherit Global Categories	☑
View Status	public ▼
Description	Automatic crash reports for "Your Software"

Creating new project for bug reports

2. Create category for the projects. If you don't need category - create something like "Auto-report" category. If you already have category - assign it. If you have master-project - you can copy categories from it.

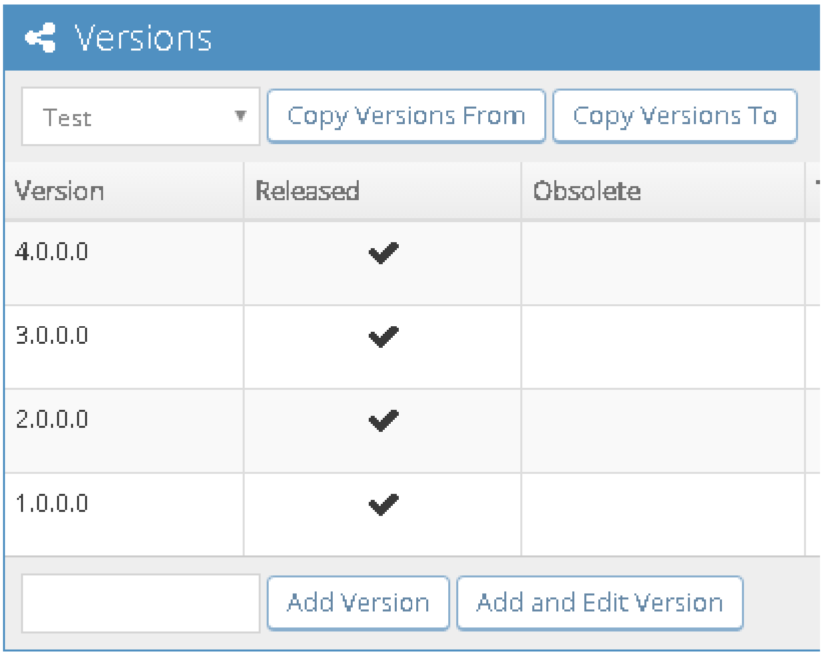
3. Create versions for the project. If you don't use versioning (highly unrecommended) - you can skip this step. If you have master-project - you can copy versions from it.

You should create new version for each release of your software. I.e. when you release (publish on site, send to custom, etc) "YourSoftware 1.0.0.0" - you need to create "1.0.0.0" version. When you release update: "YourSoftware 1.0.1.0" - you need to add "1.0.1.0" version.

Version strings can be arbitrary like "1", "1.0", "1.0.1", "1.0.1.0" or even "1.0.1.0 beta 3". However, it's recommended to use four-number versions with optional textual description, for example: "1.0.1.0" and "1.0.1.0 beta 3".

Note: if you don't want to edit project each time you release new version - you can create versions for the future use. I.e. when you release "YourSoftware 1.0.0.0" - you can create "1.0.0.0", "1.0.1.0", "1.0.2.0", "1.0.3.0"..."1.0.10.0" versions.

For example:



Version	Released	Obsolete
4.0.0.0	✓	
3.0.0.0	✓	
2.0.0.0	✓	
1.0.0.0	✓	

Batch-creating versions for future use

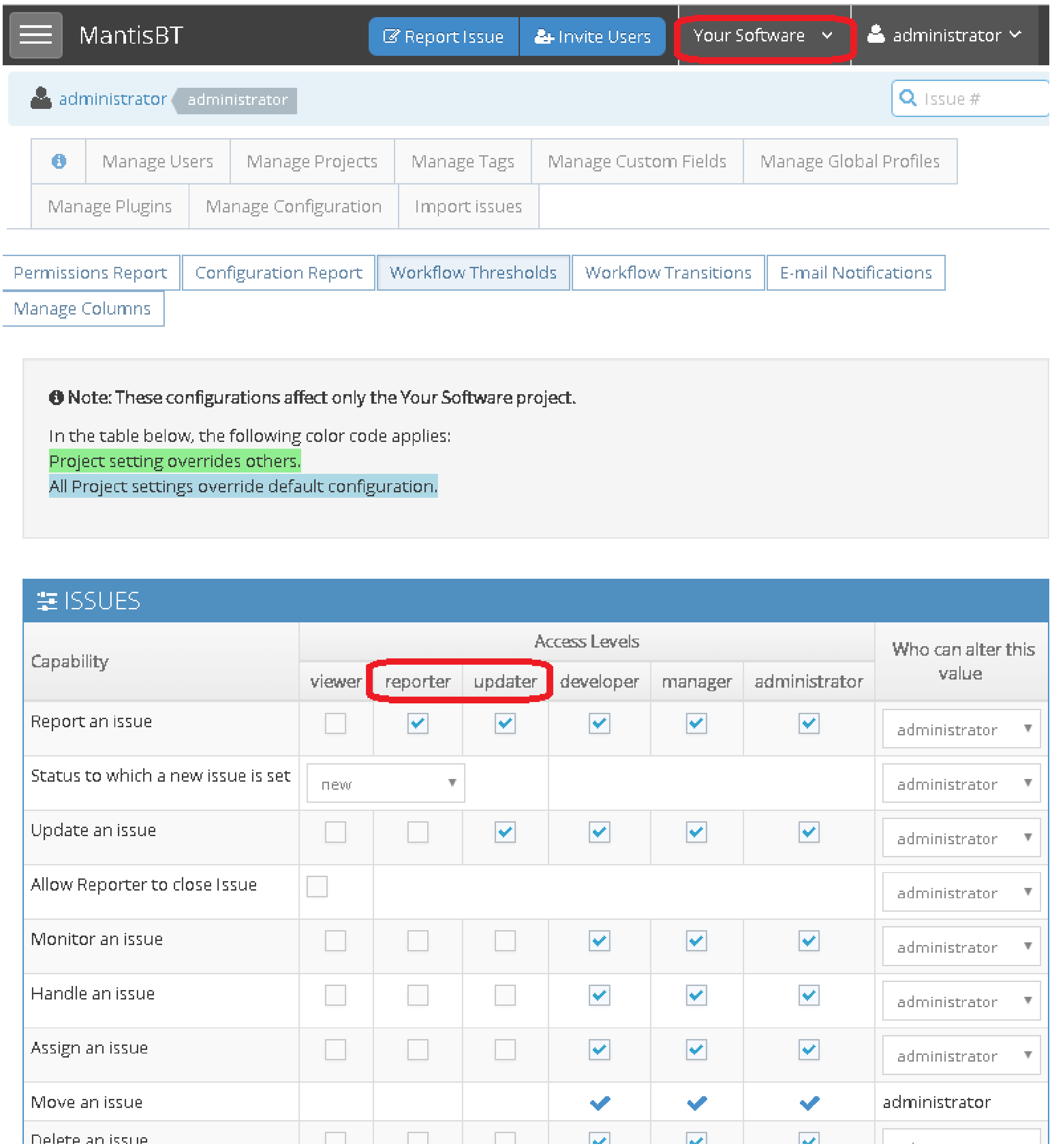
When reporting - version are taken from file's version information, so you must supply the corresponding version in description of your .exe or .dll files.

Note: if file's version information doesn't match created project versions - EurekaLog will try to use closest match. If there is nothing similar - field will be empty.

Version can be marked as "released". You should do this for all released versions of your software. Marking version as "Released" will lead to its appearance in "Product Version" field. In other words, all versions are always shown in "Target Version" field. Only released versions are shown in "Product Version" field. If you create many versions, but don't mark them as "Released" - then "Product Version" field will be empty.

Important Note: EurekaLog will look only for "Released" versions. Be sure to mark version of your .exe as "Released".

4. [Optional] (Manage/Manage Configuration/Workflow Thresholds, per-project) Setup project access rights - limit actions to minimum for "reporter" and "updater" levels:



MantisBT | Report Issue | Invite Users | Your Software | administrator

administrator administrator | Issue #

Manage Users | Manage Projects | Manage Tags | Manage Custom Fields | Manage Global Profiles | Manage Plugins | Manage Configuration | Import issues

Permissions Report | Configuration Report | Workflow Thresholds | Workflow Transitions | E-mail Notifications | Manage Columns

Note: These configurations affect only the Your Software project.

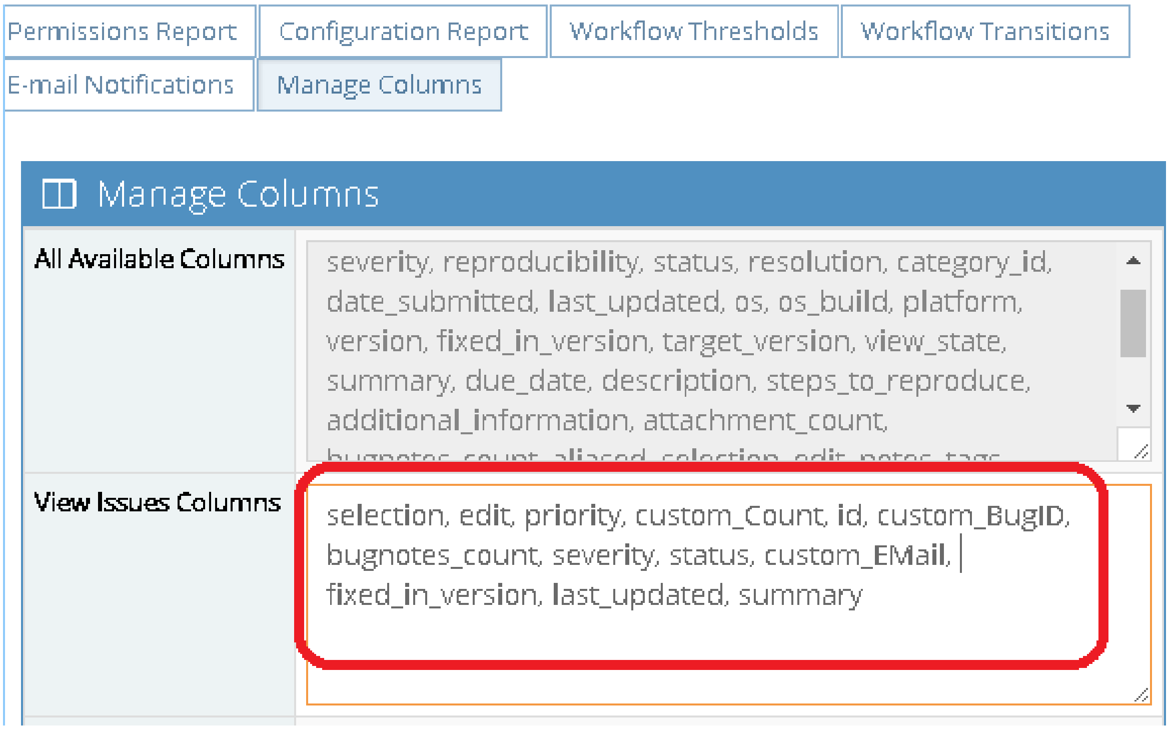
In the table below, the following color code applies:
Project setting overrides others.
All Project settings override default configuration.

ISSUES

Capability	Access Levels						Who can alter this value
	viewer	reporter	updater	developer	manager	administrator	
Report an issue	□	☑	☑	☑	☑	☑	administrator
Status to which a new issue is set	new						administrator
Update an issue	□	□	☑	☑	☑	☑	administrator
Allow Reporter to close Issue	□						administrator
Monitor an issue	□	□	□	☑	☑	☑	administrator
Handle an issue	□	□	□	☑	☑	☑	administrator
Assign an issue	□	□	□	☑	☑	☑	administrator
Move an issue				✓	✓	✓	administrator
Delete an issue	□	□	□	☑	☑	☑	

Changing rights configuration

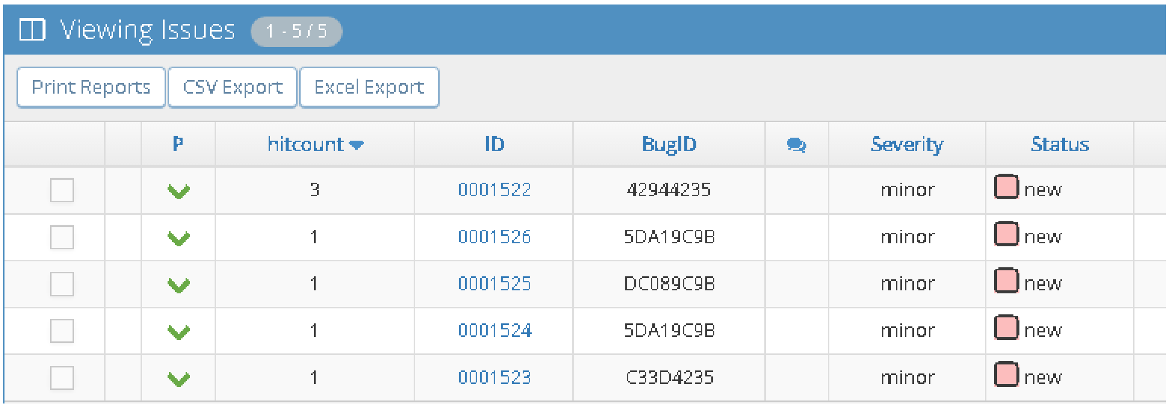
5. [Optional] (Manage/Manage Configuration/Manage Columns, per-project/per-user) Setup columns for better information view:



All Available Columns: severity, reproducibility, status, resolution, category_id, date_submitted, last_updated, os, os_build, platform, version, fixed_in_version, target_version, view_state, summary, due_date, description, steps_to_reproduce, additional_information, attachment_count, bugnotes_count, aliased_selection, edit, notes, tags

View Issues Columns: selection, edit, priority, custom_Count, id, custom_BugID, bugnotes_count, severity, status, custom_EMail, fixed_in_version, last_updated, summary

Adding custom fields to columns



		P	hitcount ▼	ID	BugID	💬	Severity	Status
□		✓	3	0001522	42944235		minor	□ new
□		✓	1	0001526	5DA19C9B		minor	□ new
□		✓	1	0001525	DC089C9B		minor	□ new
□		✓	1	0001524	5DA19C9B		minor	□ new
□		✓	1	0001523	C33D4235		minor	□ new

Example of customized columns in "View issues"

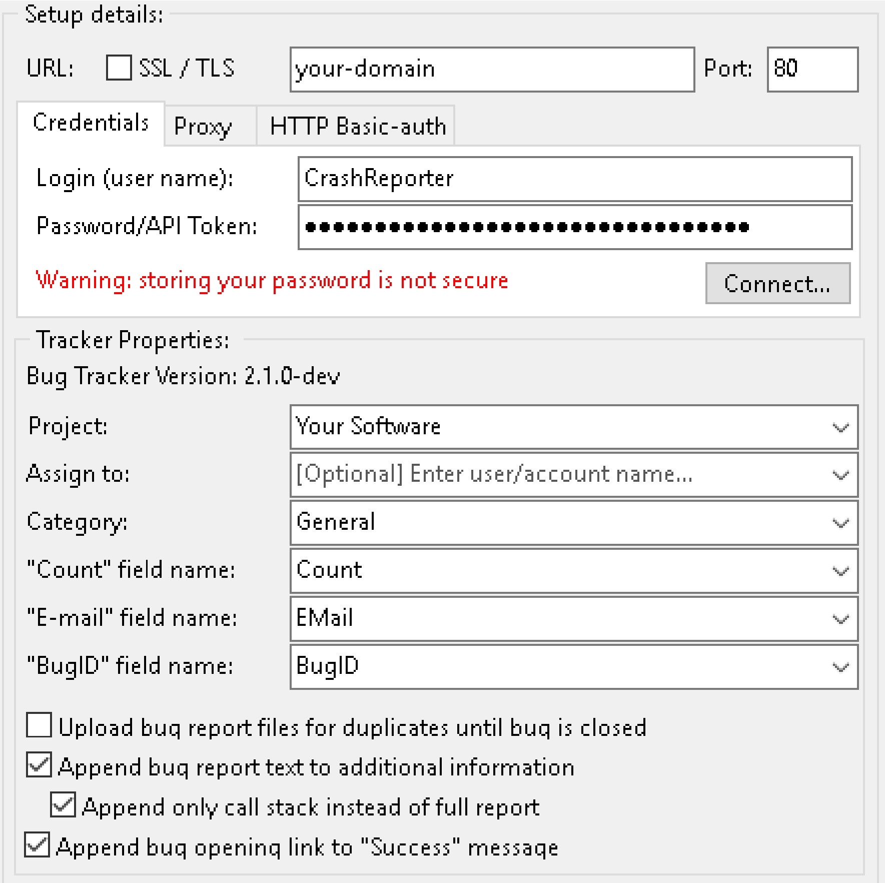
This example above uses the following column list (custom fields are taken from example of custom fields above):

> selection, edit, priority, custom_Count, id, custom_BugID, bugnotes_count, severity, status, ...

Most important field is "Count" (if you've created it) - sorting by "Count" will show you "bugs TOP 5", i.e. most popular bugs should be fixed first.

EurekaLog setup

1. Enter Mantis details into EurekaLog configuration of your projects:



Setup details:

URL: □ SSL / TLS your-domain Port: 80

Login (user name): CrashReporter

Password/API Token: ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

Warning: storing your password is not secure

Tracker Properties:

Bug Tracker Version: 2.1.0-dev

Project: Your Software

Assign to: [Optional] Enter user/account name...

Category: General

"Count" field name: Count

"E-mail" field name: EMail

"BugID" field name: BugID

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to additional information

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

Mantis settings filled into EurekaLog options

Important Note: we recommend to use API tokens when possible. Please note, that you have to enter your login (username) even if you are using API token. In any way, always use your username as login, do not use your e-mail.

2. Set any additional/common send options 304.

3. Set/fill custom fields. EurekaLog has support for automatic managing of "Count", "BugID", "E-Mail" fields. You just need to enter field names in EurekaLog options. For other custom fields you need to fill them manually, for example:

```
uses
  EEvents, ESysInfo;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe
begin
  AWebFields.Values['License'] := GetYourApplicationLicense;
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

4. Add any custom data, additional attached files, write necessary event handlers, set exception filters, etc, etc.

Testing

1. Test sending. You can do this right in the EurekaLog send options dialog 302 - by clicking on "Test..." button. This will send test bug report.

Suggested actions are:

1. Click on "Test..." button to test sending and creating of a new bug issue in Mantis.
2. Resolve any found issues (access denied, wrong values in fields, etc).
3. Once successful and there is new issue in Mantis - click on "Test..." button again. This should test updating project.
4. Resolve any found issues (access denied, etc).
5. Once successful - close existing test issue in Mantis (as "Closed" or "Resolved"). Optionally - add a note with special tags (see customizing feedback 155).
6. Click on "Test..." button again. This should test sending old (already fixed) bugs.
7. Ensure there is no error messages, no problems. You should get "success, this bug is fixed" kind of behaviour. Exact behaviour depends on your settings.
8. Delete test issue in Mantis after testing.

These actions should test that sending is actually working.

2. Now it's time to test your application-specific sending.

    1. Place debug code in your application to raise test exception and cause a test leak (if you've enabled leaks collecting).
    2. Run your application and invoke this test code.
    3. Let application crash and process bug (show dialog, send bug report, etc).
    4. Ensure that behaviour is expected.
    5. Ensure that you get all files and additional information in Mantis.
    6. Remove test code from your application.

Now your application is ready for deployment.

Maintaining projects

1. You need to create or update project versions when you ship new release of your software. If you've created a batch of versions in Mantis for future use - you may skip it until you've run out of versions.

See also:

- Managing bug reports in issue tracker 105
- Security Considerations 158
- Customizing feedback 155

7.2.1.3 BugZilla setup

This article is part of Managing bug report in issue tracker 105 series.

See managing bug reports in issue tracker 105 for common information. Please, **read it first**. For common information and setup of BugZilla itself - please see this article 407. The text below assumes that you already completed BugZilla installation.

Below are detailed steps for recommended BugZilla setup for automatic bug report submission. Before going through setup - make sure to upgrade your BugZilla to the latest version.

Some steps below are optional, some steps must be executed only once (like custom fields creation), other are executed from time to time (like creating new projects for your new products) and the rest are executed regularly (like creating product versions).

Full list of necessary actions contains:

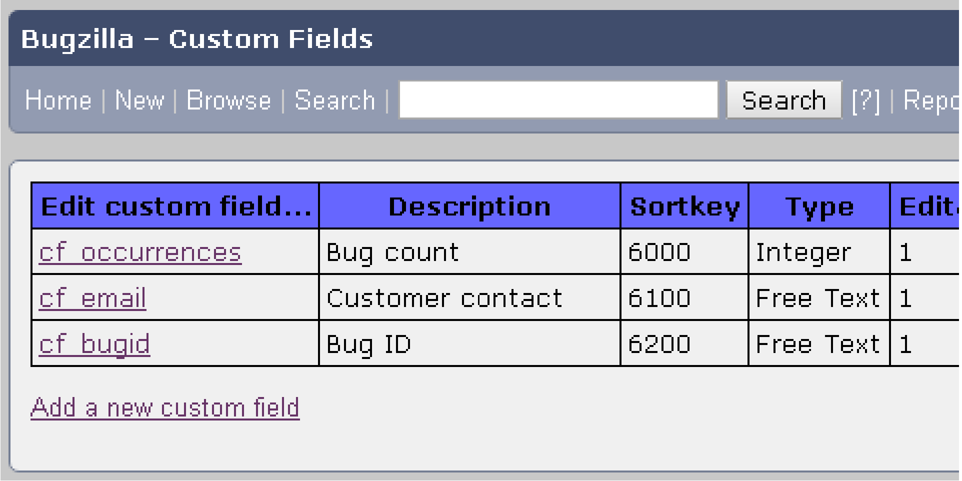
1. Creating custom fields (single act)
2. Creating user accounts (single act or per product)
3. Creating projects and setting it up (single act or per product)
4. EurekaLog setup (per product)
5. Testing (as required)
6. Maintaining project (regularly or from time to time)

Please note that all actions below are just examples. It's recommendation, but it's not necessary to be absolutely like that. You may use another configuration.

Creating custom fields

1. (Administration/Custom Fields) Create custom field to improve usefulness of EurekaLog. Most important field is "Count" - to store number of sent/occurred problems. Its type should be "Integer"; field name can be arbitrary, like "Occurrences", "Bug count", "Popularity", "Incidents", "Hit Count", etc. Other suggested custom fields are: "BugID" (to store BugID and search issues) and "e-mail" ("user e-mail", which is typically entered in MS Classic error dialog 377). Both fields should be text fields. Do not use "Bug ID" field type for "BugID" field. Again, field names can be anything.

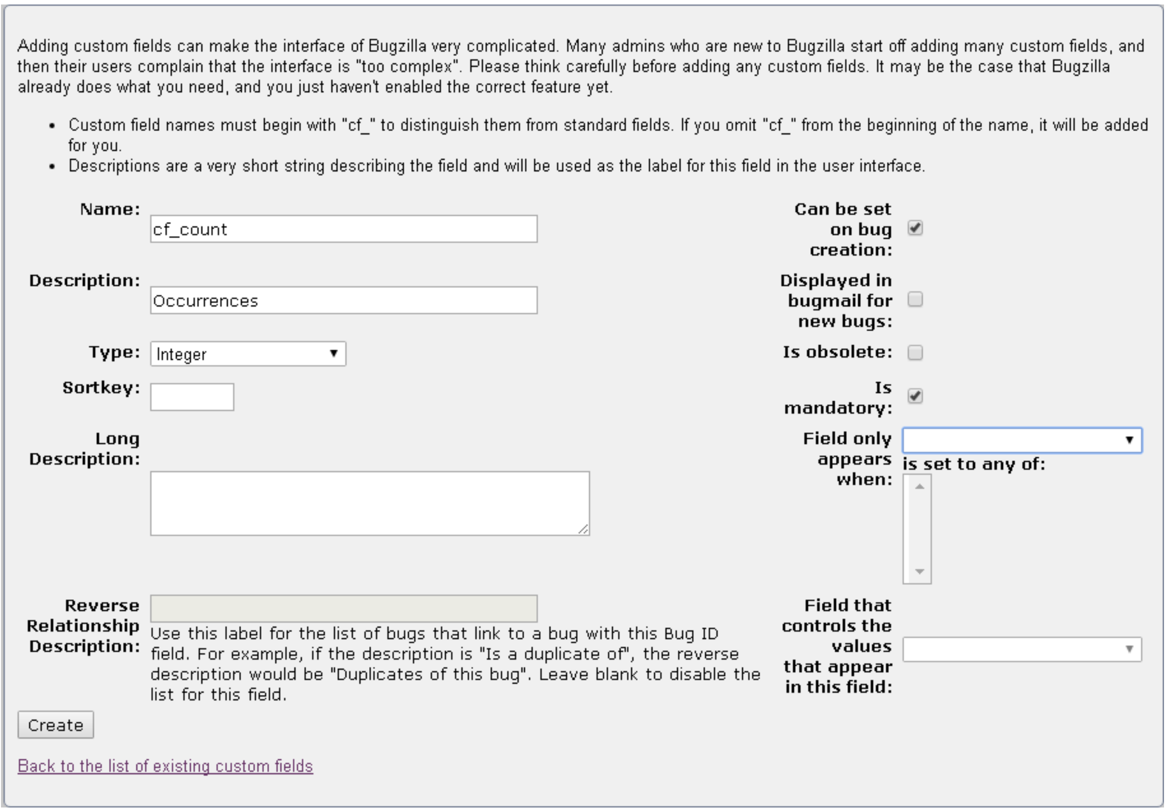
We strongly recommend to create at least "Count" field.



Edit custom field...	Description	Sortkey	Type	Edit
cf_occurrences	Bug count	6000	Integer	1
cf_email	Customer contact	6100	Free Text	1
cf_bugid	Bug ID	6200	Free Text	1

Add a new custom field

Example: two new custom fields



Adding custom fields can make the interface of Bugzilla very complicated. Many admins who are new to Bugzilla start off adding many custom fields, and then their users complain that the interface is "too complex". Please think carefully before adding any custom fields. It may be the case that Bugzilla already does what you need, and you just haven't enabled the correct feature yet.

- Custom field names must begin with "cf_" to distinguish them from standard fields. If you omit "cf_" from the beginning of the name, it will be added for you.
- Descriptions are a very short string describing the field and will be used as the label for this field in the user interface.

Name: cf_count

Description: Occurrences

Type: Integer

Sortkey:

Long Description:

Reverse Relationship Description: Use this label for the list of bugs that link to a bug with this Bug ID field. For example, if the description is "Is a duplicate of", the reverse description would be "Duplicates of this bug". Leave blank to disable the list for this field.

Can be set on bug creation: ☑

Displayed in bugmail for new bugs: ☐

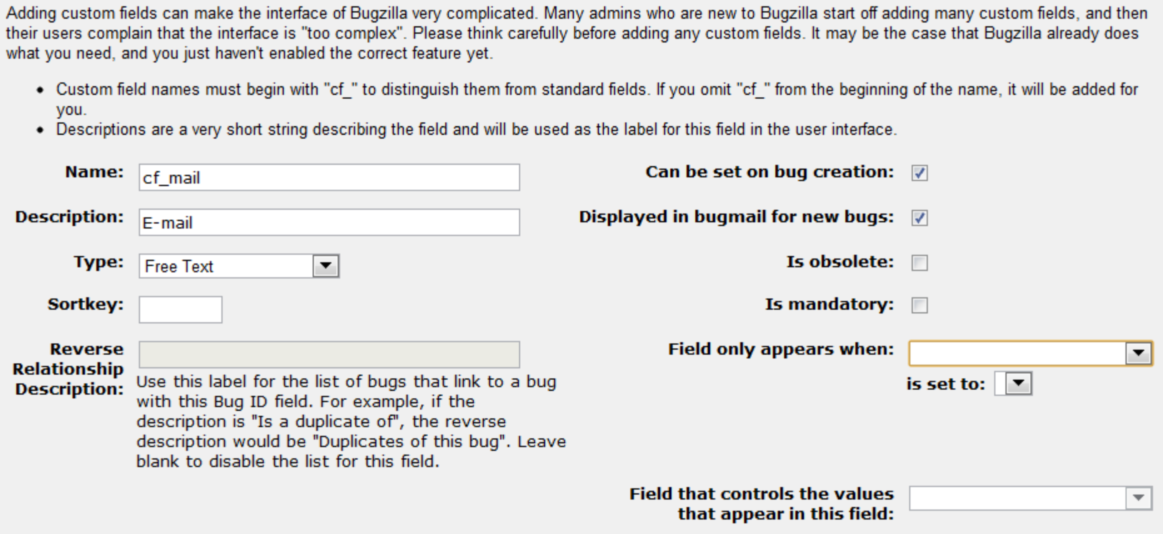
Is obsolete: ☐

Is mandatory: ☑

Field only appears when: is set to any of:

Field that controls the values that appear in this field:

Suggested setup for "Count" field



Adding custom fields can make the interface of Bugzilla very complicated. Many admins who are new to Bugzilla start off adding many custom fields, and then their users complain that the interface is "too complex". Please think carefully before adding any custom fields. It may be the case that Bugzilla already does what you need, and you just haven't enabled the correct feature yet.

- Custom field names must begin with "cf_" to distinguish them from standard fields. If you omit "cf_" from the beginning of the name, it will be added for you.
- Descriptions are a very short string describing the field and will be used as the label for this field in the user interface.

Name: cf_mail

Description: E-mail

Type: Free Text

Sortkey:

Reverse Relationship Description:
Use this label for the list of bugs that link to a bug with this Bug ID field. For example, if the description is "Is a duplicate of", the reverse description would be "Duplicates of this bug". Leave blank to disable the list for this field.

Can be set on bug creation: ☑

Displayed in bugmail for new bugs: ☑

Is obsolete: □

Is mandatory: □

Field only appears when:

is set to:

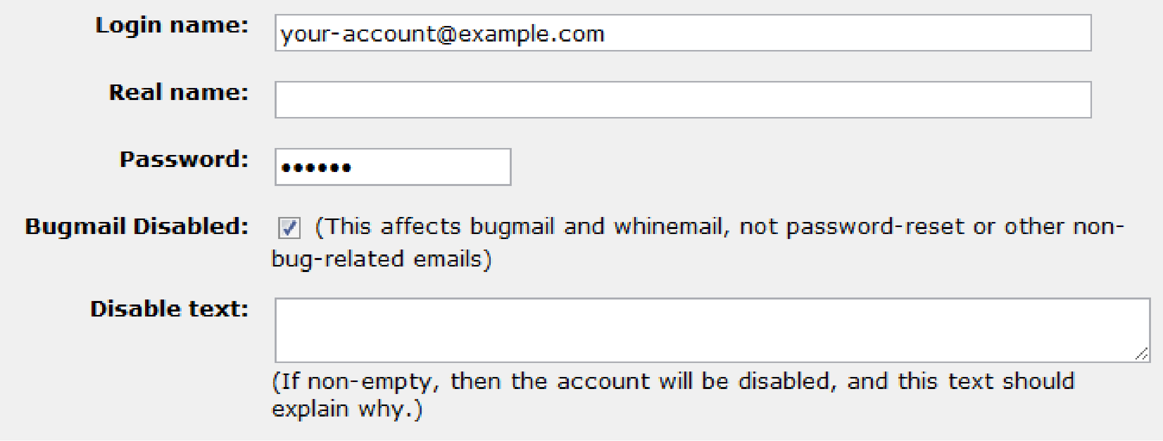
Field that controls the values that appear in this field:

Suggested setup for "e-mail" and "BugID" fields

2. Create any other additional fields as you need/like (you can submit values for custom fields at run-time via OnCustomWebFieldsRequest event).

Creating user accounts

1. (Administration/Users) Create new use account for bug report submission. Create it with disabled e-mail notifications:



Login name: your-account@example.com

Real name:

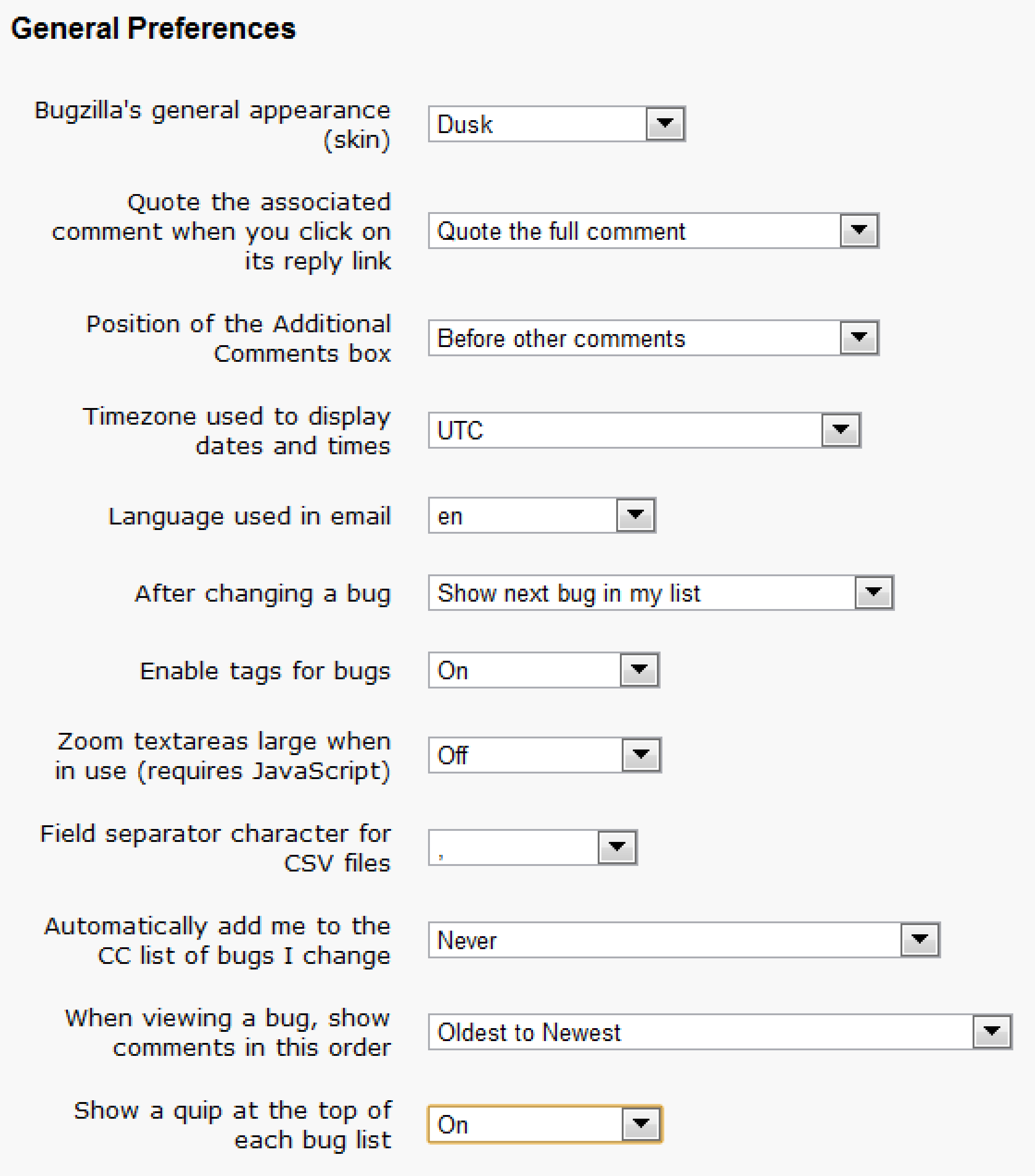
Password: ••••••

Bugmail Disabled: ☑ (This affects bugmail and whinemail, not password-reset or other non-bug-related emails)

Disable text:
(If non-empty, then the account will be disabled, and this text should explain why.)

Creating new user

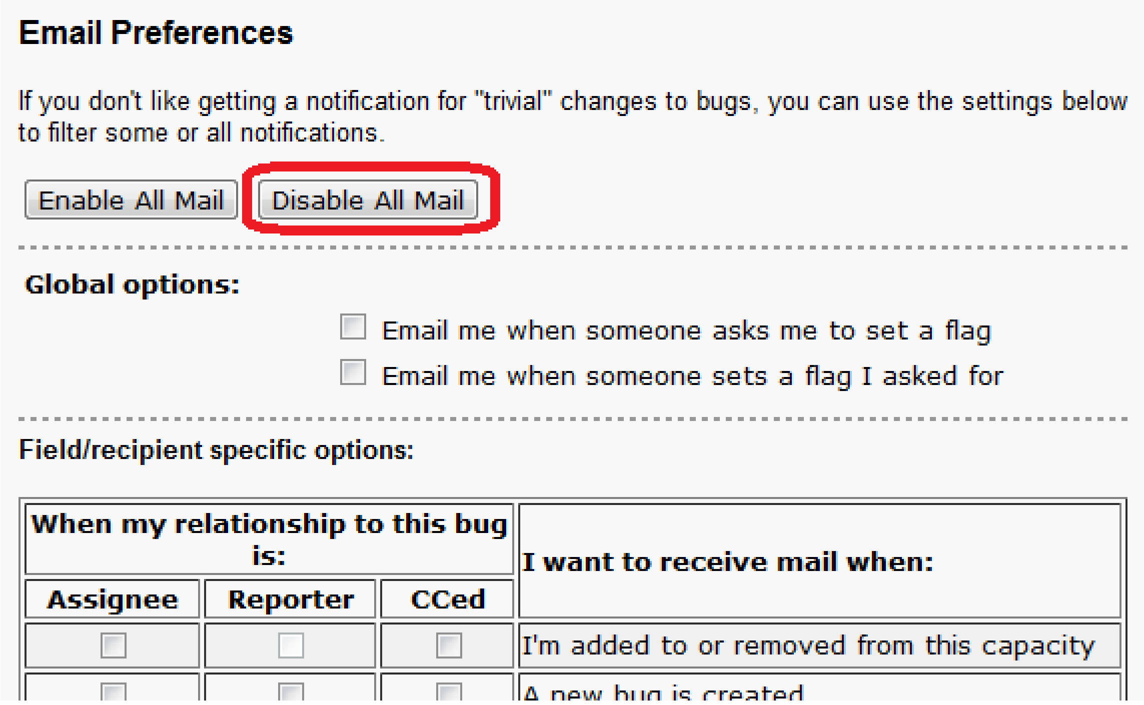
2. Log off and log in as this user. Go to "Preferences" and clear e-mail notifications, set default settings and switch language to "English":



General Preferences

Bugzilla's general appearance (skin)	Dusk
Quote the associated comment when you click on its reply link	Quote the full comment
Position of the Additional Comments box	Before other comments
Timezone used to display dates and times	UTC
Language used in email	en
After changing a bug	Show next bug in my list
Enable tags for bugs	On
Zoom textareas large when in use (requires JavaScript)	Off
Field separator character for CSV files	,
Automatically add me to the CC list of bugs I change	Never
When viewing a bug, show comments in this order	Oldest to Newest
Show a quip at the top of each bug list	On

Setting properties of submitter account - general



Email Preferences

If you don't like getting a notification for "trivial" changes to bugs, you can use the settings below to filter some or all notifications.

Global options:

□ Email me when someone asks me to set a flag

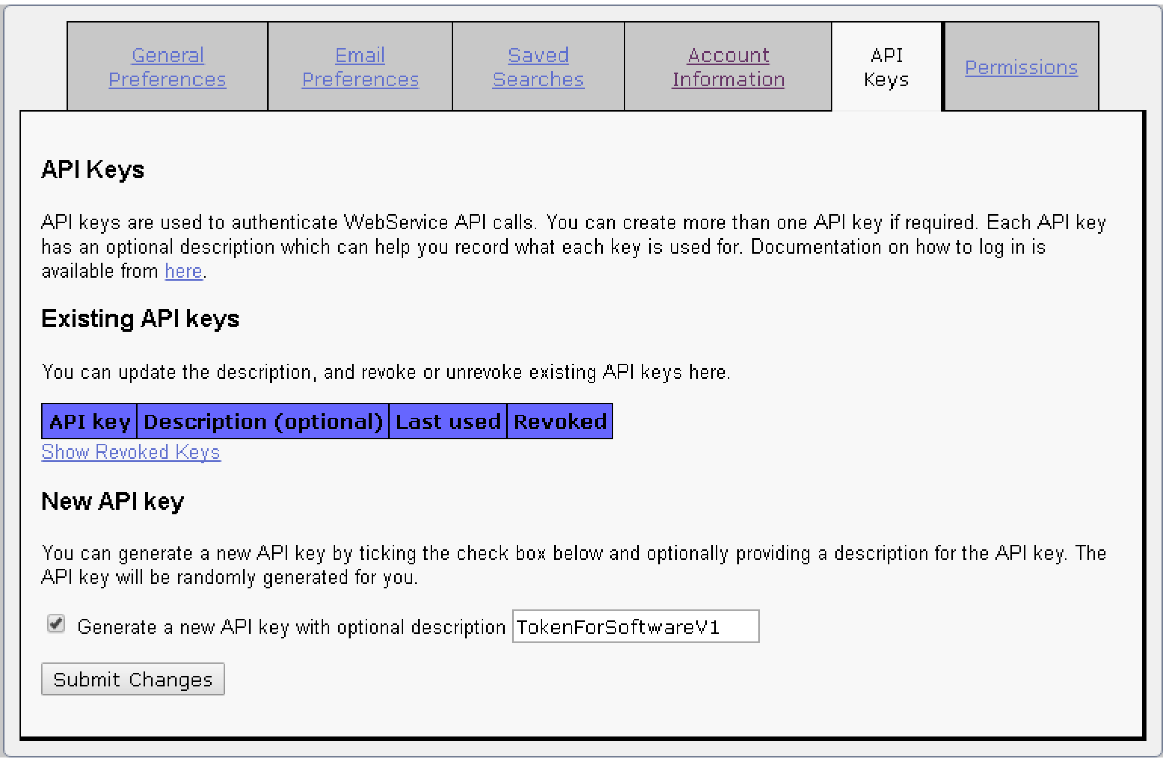
□ Email me when someone sets a flag I asked for

Field/recipient specific options:

When my relationship to this bug is:			I want to receive mail when:
Assignee	Reporter	CCed	
□	□	□	I'm added to or removed from this capacity
□	□	□	A new bug is created

Setting properties of submitter account - e-mail notifications

3. (Optional, but strongly recommended; only for latest BugZilla versions) Go to Preferences / API Keys and create new API token:



API Keys

API keys are used to authenticate WebService API calls. You can create more than one API key if required. Each API key has an optional description which can help you record what each key is used for. Documentation on how to log in is available from here.

Existing API keys

You can update the description, and revoke or unrevoke existing API keys here.

API key	Description (optional)	Last used	Revoked

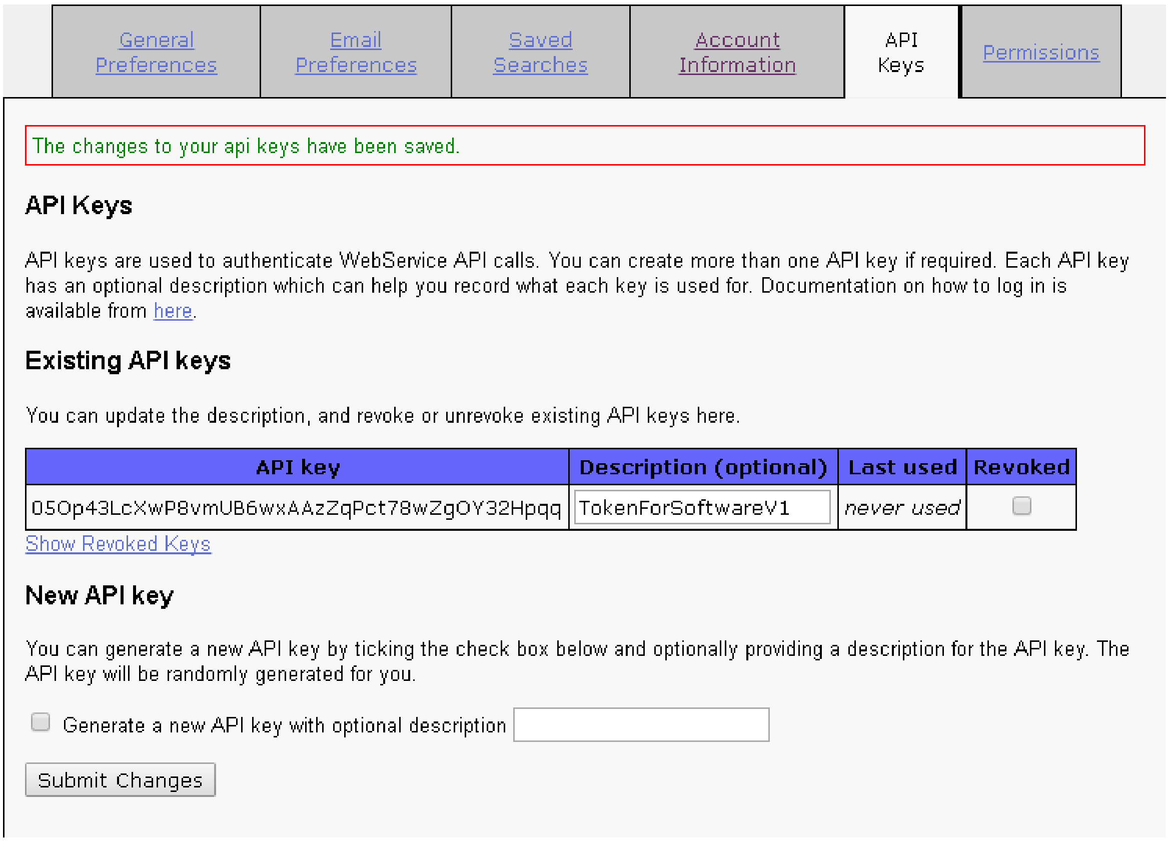
Show Revoked Keys

New API key

You can generate a new API key by ticking the check box below and optionally providing a description for the API key. The API key will be randomly generated for you.

☑ Generate a new API key with optional description TokenForSoftwareV1

Creating new API token for bug reporter account



The changes to your api keys have been saved.

API Keys

API keys are used to authenticate WebService API calls. You can create more than one API key if required. Each API key has an optional description which can help you record what each key is used for. Documentation on how to log in is available from here.

Existing API keys

You can update the description, and revoke or unrevoke existing API keys here.

API key	Description (optional)	Last used	Revoked
05Op43LcXwP8vmUB6wxAAzZqPct78wZgOY32Hpqq	TokenForSoftwareV1	*never used*	□

Show Revoked Keys

New API key

You can generate a new API key by ticking the check box below and optionally providing a description for the API key. The API key will be randomly generated for you.

□ Generate a new API key with optional description

New API key was created

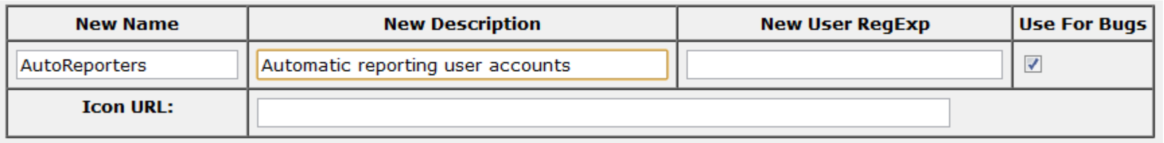
Once API key was created - select it and copy to buffer. You will need to enter it into EurekaLog configuration later.

You may create additional token keys.

4. Now, log off and log in again as administrator.

Repeat these steps for each bug submitter user account which you've created.

5. You can also create a new group of users and include all bug-reporting user accounts into that group (Administration/Groups):

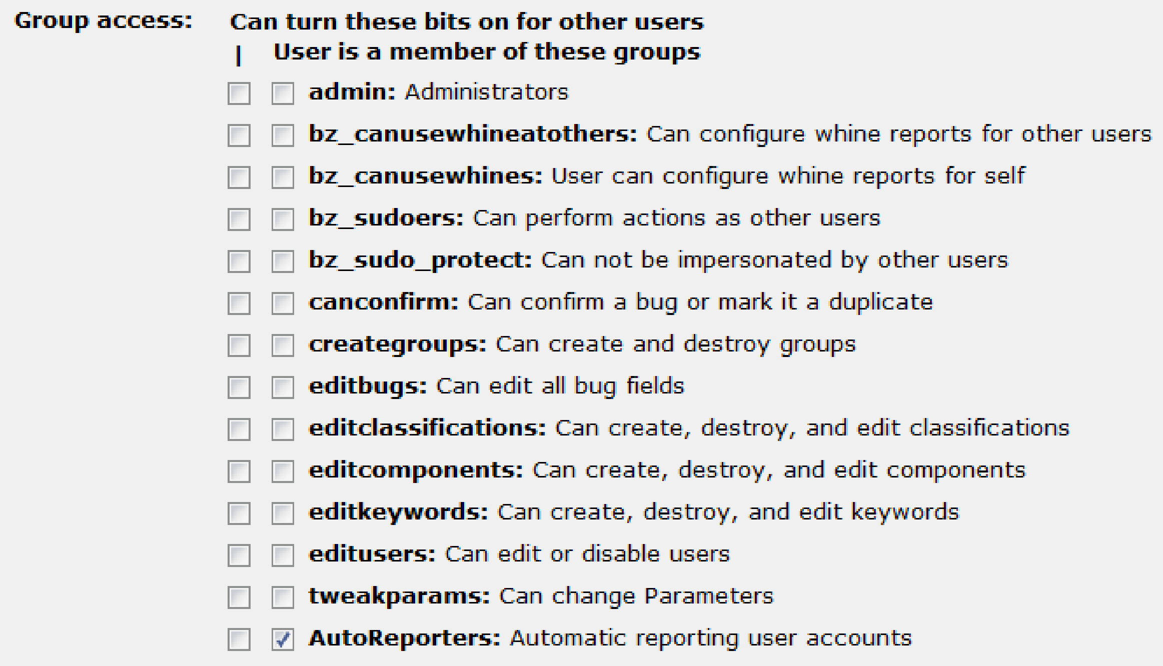


New Name	New Description	New User RegExp	Use For Bugs
AutoReporters	Automatic reporting user accounts		☑
Icon URL:			

Creating new user group for bug reporting

Note: by default "editbugs" group is assigned to all users. You may want to exclude your bug reporting account from "editbugs" - to do this, edit "editbugs" group and remove default regular expression ".*". Include all necessary users to that group.

6. Open bug reporting user account and include it into a bug reporting group:



Group access: **Can turn these bits on for other users**
| **User is a member of these groups**

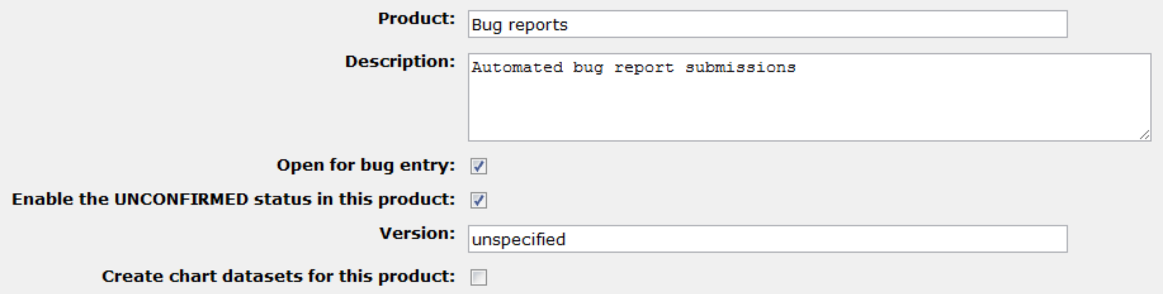
□ □ **admin:** Administrators
□ □ **bz_canusewhineatothers:** Can configure whine reports for other users
□ □ **bz_canusewhines:** User can configure whine reports for self
□ □ **bz_sudoers:** Can perform actions as other users
□ □ **bz_sudo_protect:** Can not be impersonated by other users
□ □ **canconfirm:** Can confirm a bug or mark it a duplicate
□ □ **creategroups:** Can create and destroy groups
□ □ **editbugs:** Can edit all bug fields
□ □ **editclassifications:** Can create, destroy, and edit classifications
□ □ **editcomponents:** Can create, destroy, and edit components
□ □ **editkeywords:** Can create, destroy, and edit keywords
□ □ **editusers:** Can edit or disable users
□ □ **tweakparams:** Can change Parameters
□ ☑ **AutoReporters:** Automatic reporting user accounts

Suggested group setup for bug reporting user accounts

Repeat this for all bug reporting accounts.

Creating projects

1. (Administration/Products) Create project for your software product (projects are called products in BugZilla). You may create several projects - one for each of your software products.



Product: Bug reports
Description: Automated bug report submissions
Open for bug entry: ☑
Enable the UNCONFIRMED status in this product: ☑
Version: unspecified
Create chart datasets for this product: □

Creating new project for bug reports

2. Create components for the projects. If you don't need components - create something like "General" or "unspecified" component. Typically component is used for identification of the part of your software product.

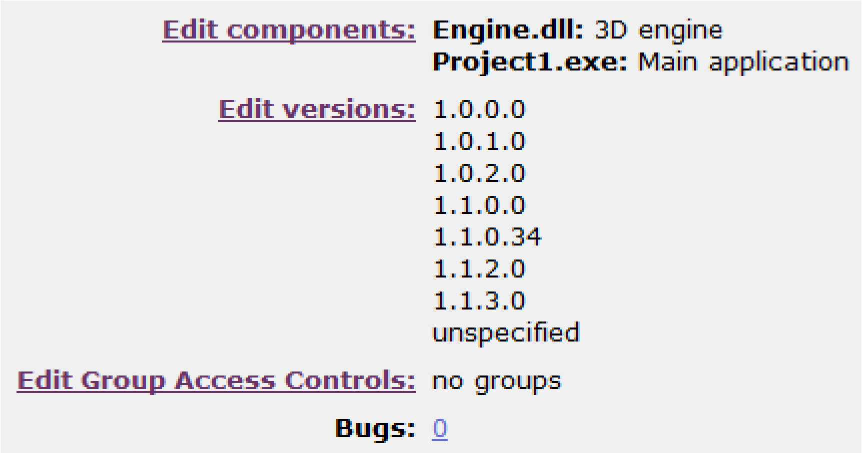
3. Create versions for the project. If you don't use versioning (highly unrecommended) - you can skip this step.

You should create new version for each release of your software. I.e. when you release (publish on site, send to custom, etc) "YourSoftware 1.0.0.0" - you need to create "1.0.0.0" version. When you release update: "YourSoftware 1.0.1.0" - you need to add "1.0.1.0" version.

Version strings can be arbitrary like "1", "1.0", "1.0.1", "1.0.1.0" or even "1.0.1.0 beta 3". However, it's recommended to use four-number versions with optional textual description, for example: "1.0.1.0" and "1.0.1.0 beta 3".

Note: if you don't want to edit project each time you release new version - you can create versions for the future use. I.e. when you release "YourSoftware 1.0.0.0" - you can create "1.0.0.0", "1.0.1.0", "1.0.2.0", "1.0.3.0"..."1.0.10.0" versions.

For example:

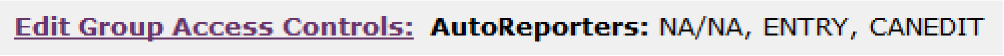


Edit components:	**Engine.dll:** 3D engine **Project1.exe:** Main application
Edit versions:	1.0.0.0 1.0.1.0 1.0.2.0 1.1.0.0 1.1.0.34 1.1.2.0 1.1.3.0 unspecified
Edit Group Access Controls:	no groups
Bugs:	0

Components and versions

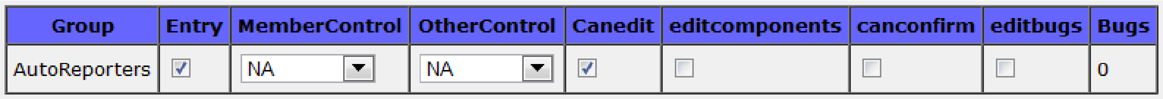
When reporting - version are taken from file's version information, so you must supply the corresponding version in description of your .exe or .dll files.

4. Setup project access rights - limit actions to members of auto-reporting group:



Edit Group Access Controls: **AutoReporters:** NA/NA, ENTRY, CANEDIT

Example of limited rights configuration



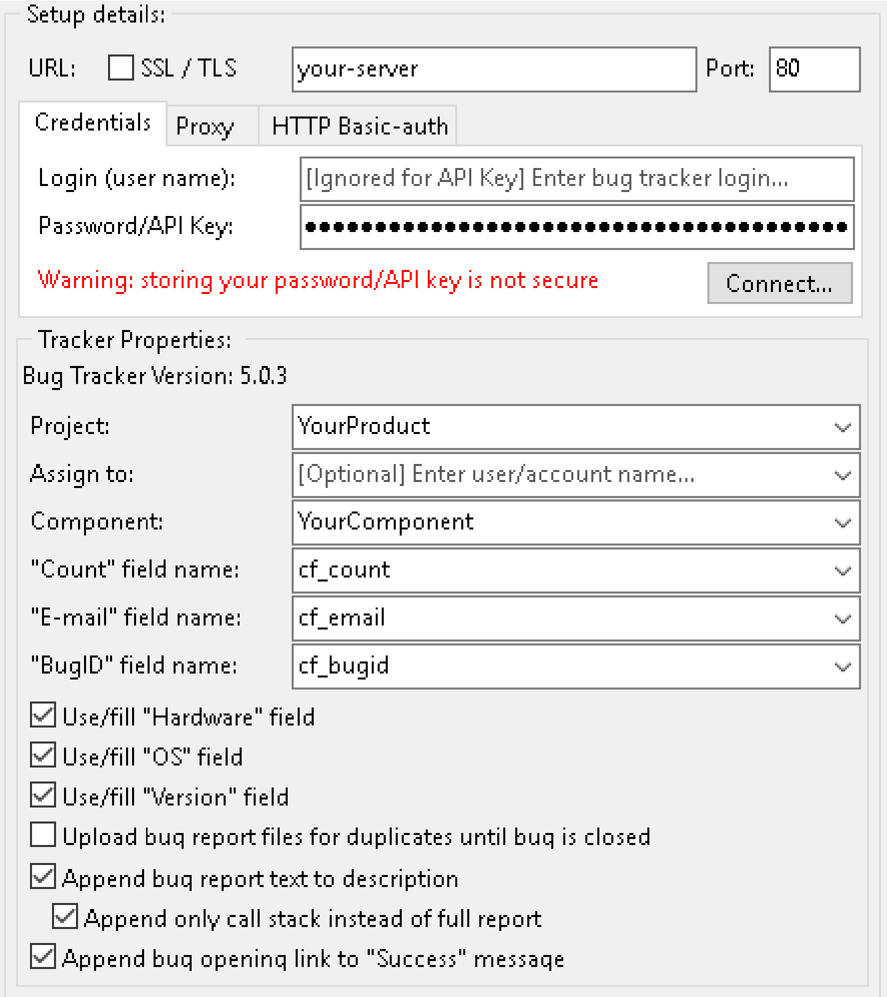
Group	Entry	MemberControl	OtherControl	Canedit	editcomponents	canconfirm	editbugs	Bugs
AutoReporters	☑	NA	NA	☑	□	□	□	0

Detailed view of group access rights to the project

Note: by default bug reporting user accounts will have access to all other projects, unless these projects has group with "Entry" assigned. I.e. by default project has no group assigned - this means world access to project. You should explicitly add group to project to exclude other users (non-members) from accessing project.

EurekaLog setup

1. Enter BugZilla details into EurekaLog configuration of your projects:



Setup details:

URL: □ SSL / TLS your-server Port: 80

Login (user name): [Ignored for API Key] Enter bug tracker login...

Password/API Key: ●●

Warning: storing your password/API key is not secure

Tracker Properties:

Bug Tracker Version: 5.0.3

Project:	YourProduct
Assign to:	[Optional] Enter user/account name...
Component:	YourComponent
"Count" field name:	cf_count
"E-mail" field name:	cf_email
"BugID" field name:	cf_bugid

☑ Use/fill "Hardware" field
☑ Use/fill "OS" field
☑ Use/fill "Version" field
□ Upload bug report files for duplicates until bug is closed
☑ Append bug report text to description
☑ Append only call stack instead of full report
☑ Append bug opening link to "Success" message

BugZilla settings filled into EurekaLog options

Important Note: we recommend to use API keys (tokens) when possible. If you are still going to use login/password pair (for example, you are using old BugZilla version, which does not have API keys) - use your e-mail as login.

2. Set any additional/common send options 304.

3. Set/fill custom fields. EurekaLog has support for automatic managing of "Count", "BugID" and "E-Mail" fields. You just need to enter field names in EurekaLog options. For other custom fields you have to fill them manually, for example:

```
uses
  EEvents, ESysInfo;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe
begin
  AWebFields.Values['cf_license'] := GetYourApplicationLicense;
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

4. Add any custom data, additional attached files, write necessary event handlers, set exception filters, etc, etc.

Testing

1. Test sending. You can do this right in the EurekaLog send options dialog 302 - by clicking on "Test..." button. This will send test bug report.

Suggested actions are:

1. Click on "Test..." button to test sending and creating of a new bug issue in BugZilla.
2. Resolve any found issues (access denied, wrong values in fields, etc).
3. Once successful and there is new issue in BugZilla - click on "Test..." button again. This should test updating project.
4. Resolve any found issues (access denied, etc).
5. Once successful - close existing test issue in BugZilla (as "RESOLVED"). Optionally - add a note with special tags (see customizing feedback 155).
6. Click on "Test..." button again. This should test sending old (already fixed) bugs.
7. Ensure there is no error messages, no problems. You should get "success, this bug is fixed" kind of behaviour. Exact behaviour depends on your settings.
8. Delete test issue in BugZilla after testing.

These actions should test that sending is actually working.

2. Now it's time to test your application-specific sending.

1. Place debug code in your application to raise test exception and cause a test leak (if you've enabled leaks collecting).
2. Run your application and invoke this test code.
3. Let application crash and process bug (show dialog, send bug report, etc).
4. Ensure that behaviour is expected.
5. Ensure that you get all files and additional information in BugZilla.
6. Remove test code from your application.

Now your application is ready for deployment.

Maintaining projects

1. You need to create or update project versions when you ship new release of your software. If you've created a batch of versions in BugZilla for future use - you may skip it until you've run out of versions.

See also:

- Managing bug reports in issue tracker 105
- Security Considerations 158
- Customizing feedback 155

7.2.1.4 JIRA setup

This article is part of Managing bug report in issue tracker 105 series.

See managing bug reports in issue tracker 105 for common information. Please, **read it first**. For common information and setup of JIRA itself - please see this article 408. The text below assumes that you already completed JIRA installation.

Below are detailed steps for recommended JIRA setup for automatic bug report submission. Before going through setup - make sure to upgrade your JIRA to the latest version.

Some steps below are optional, some steps must be executed only once (like custom fields creation), other are executed from time to time (like creating new projects for your new products) and the rest are executed regularly (like creating product versions).

Full list of necessary actions contains:
1. Creating custom fields (single act)
2. Creating user accounts (single act or per product)
3. Creating projects and setting it up (single act or per product)
4. EurekaLog setup (per product)
5. Testing (as required)

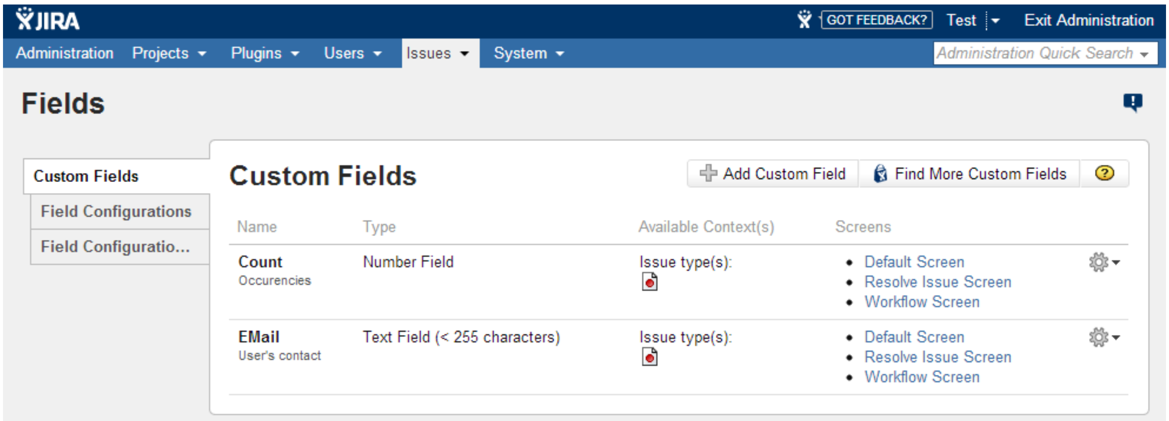
6. Maintaining project (regularly or from time to time)

Please note that all actions below are just examples. It's recommendation, but it's not necessary to be absolutely like that. You may use another configuration.

Creating custom fields

1. (Administration/Custom Fields) Create custom field to improve usefulness of EurekaLog. Most important field is "Count" - to store number of sent/occurred problems. Its type should be "Number" (integer); field name can be arbitrary, like "Occurrences", "Bug count", "Popularity", "Incidents", "Hit Count", etc. Other suggested custom fields are: "BugID" (to store BugID and search issues) and "e-mail" ("user e-mail", which is typically entered in MS Classic error dialog 377). Both fields should be text fields. Again, field names can be anything.

We strongly recommend to create at least "Count" and "BugID" fields.



Custom Fields

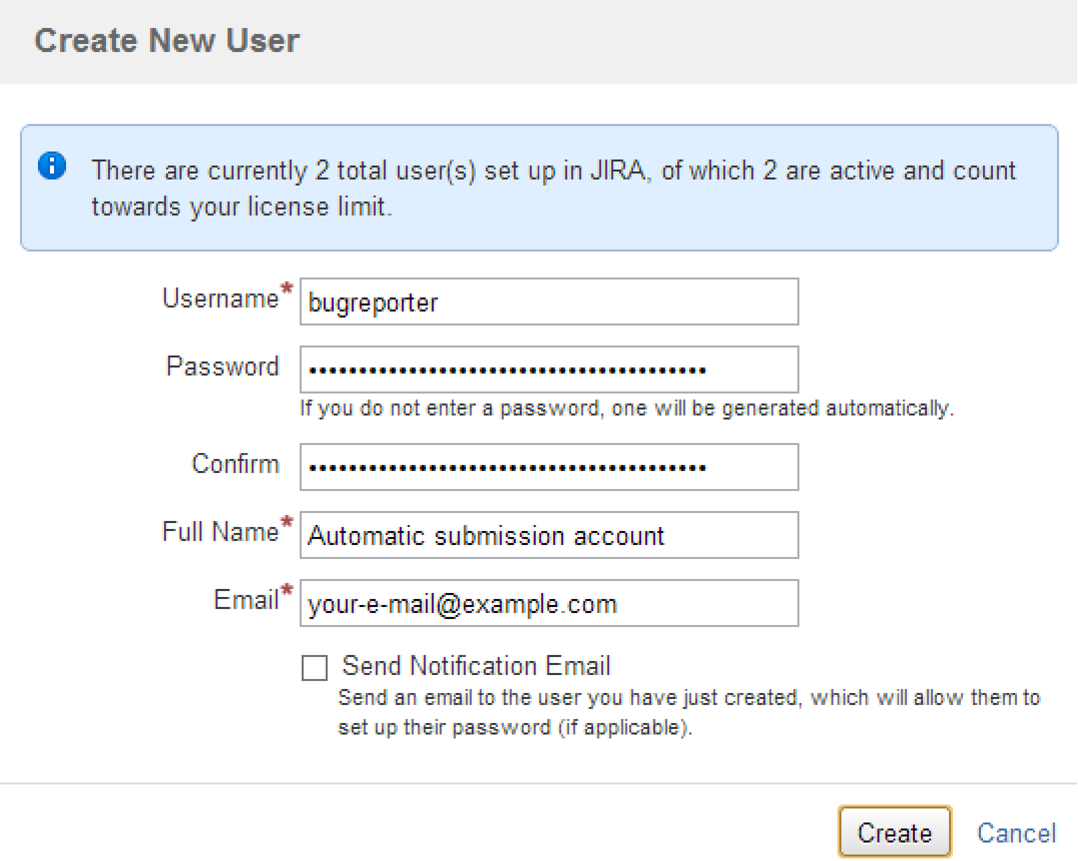
Name	Type	Available Context(s)	Screens
Count Occurencies	Number Field	Issue type(s):	Default Screen Resolve Issue Screen Workflow Screen
EMail User's contact	Text Field (< 255 characters)	Issue type(s):	Default Screen Resolve Issue Screen Workflow Screen

Example: two new custom fields

2. Create any other additional fields as you need/like (you can submit values for custom fields at run-time via OnCustomWebFieldsRequest event).

Creating user accounts

1. (Administration/Users) Create new use account for bug report submission. Create it with disabled e-mail notification:



Create New User

There are currently 2 total user(s) set up in JIRA, of which 2 are active and count towards your license limit.

Username* bugreporter

Password ••

If you do not enter a password, one will be generated automatically.

Confirm ••

Full Name* Automatic submission account

Email* your-e-mail@example.com

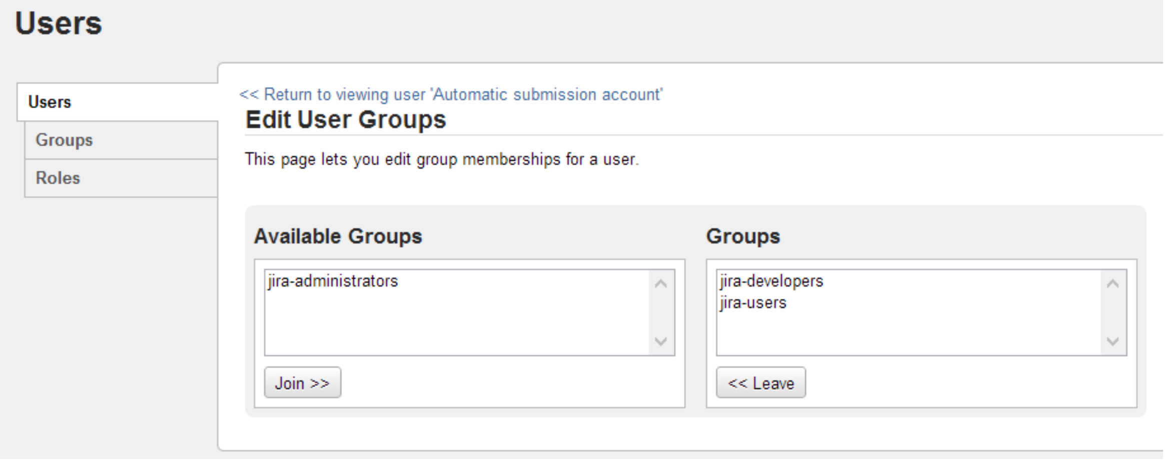
□ Send Notification Email

Send an email to the user you have just created, which will allow them to set up their password (if applicable).

Creating new user

Important Note: please remember username of new account. You will be using it later - when entering login credentials into EurekaLog. Do not use e-mail or full name.

2. Click on "Edit groups" link and add user into developers group (users group have read-only access; developers group have read-write access; administrators group have full control access):



Edit User Groups

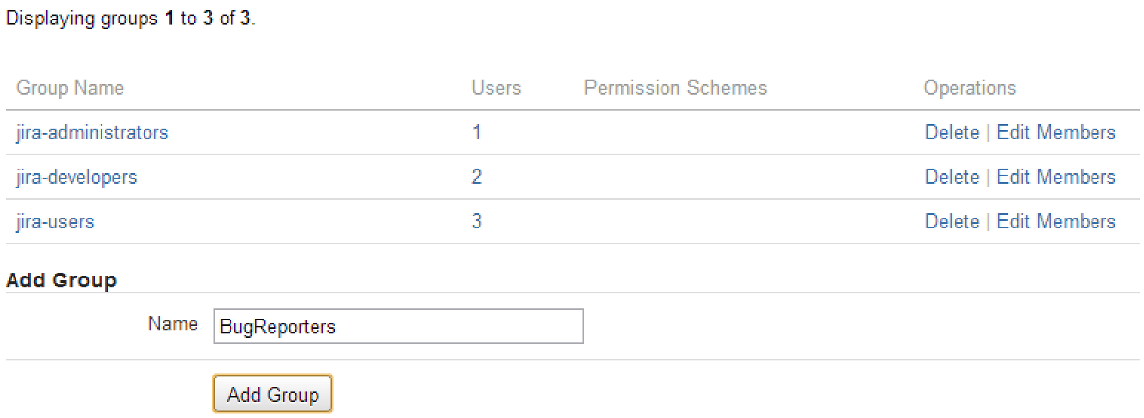
This page lets you edit group memberships for a user.

Available Groups	Groups
jira-administrators	jira-developers jira-users

Including account into groups

3. Repeat these steps for each bug submitter user account which you want to create.

4. You can also create a new group and/or role and include all bug-reporting user accounts into that group (Administration/Groups and Administration/Roles):



Displaying groups **1** to **3** of **3**.

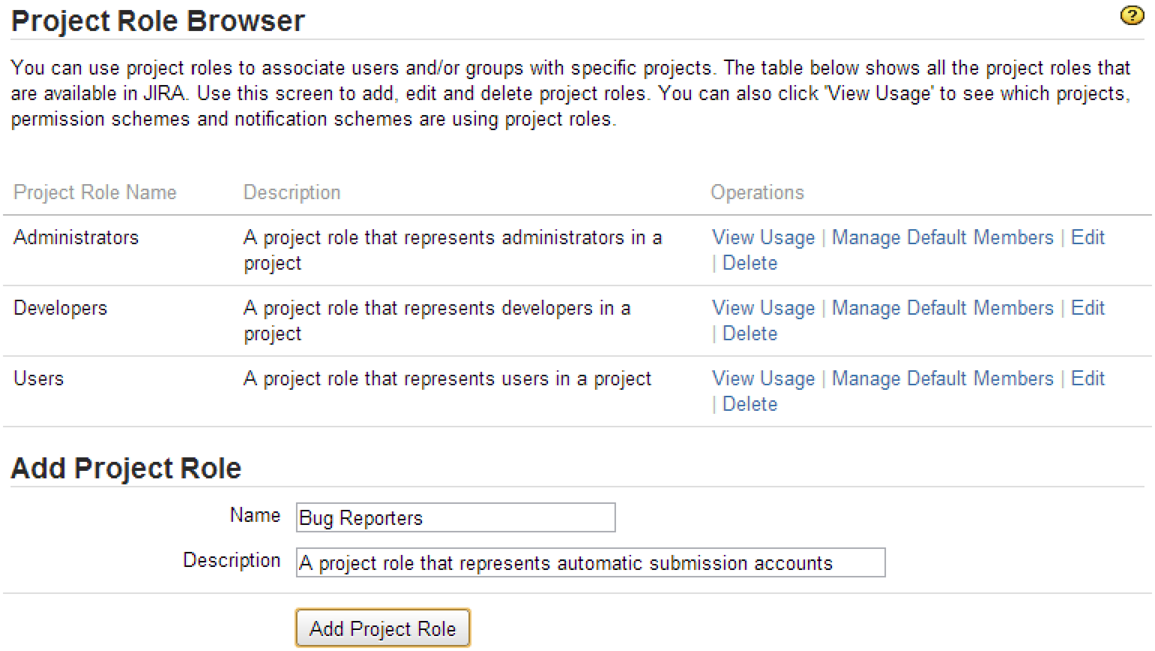
Group Name	Users	Permission Schemes	Operations
jira-administrators	1		Delete | Edit Members
jira-developers	2		Delete | Edit Members
jira-users	3		Delete | Edit Members

Add Group

Name: BugReporters

Add Group

Creating new user group for bug reporting



Project Role Browser

You can use project roles to associate users and/or groups with specific projects. The table below shows all the project roles that are available in JIRA. Use this screen to add, edit and delete project roles. You can also click 'View Usage' to see which projects, permission schemes and notification schemes are using project roles.

Project Role Name	Description	Operations
Administrators	A project role that represents administrators in a project	View Usage | Manage Default Members | Edit | Delete
Developers	A project role that represents developers in a project	View Usage | Manage Default Members | Edit | Delete
Users	A project role that represents users in a project	View Usage | Manage Default Members | Edit | Delete

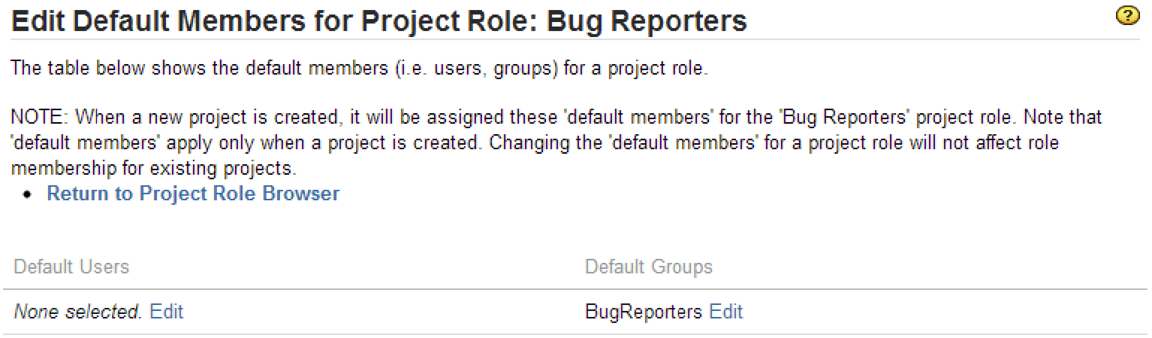
Add Project Role

Name: Bug Reporters

Description: A project role that represents automatic submission accounts

Add Project Role

Creating new role for bug reporting



Edit Default Members for Project Role: Bug Reporters

The table below shows the default members (i.e. users, groups) for a project role.

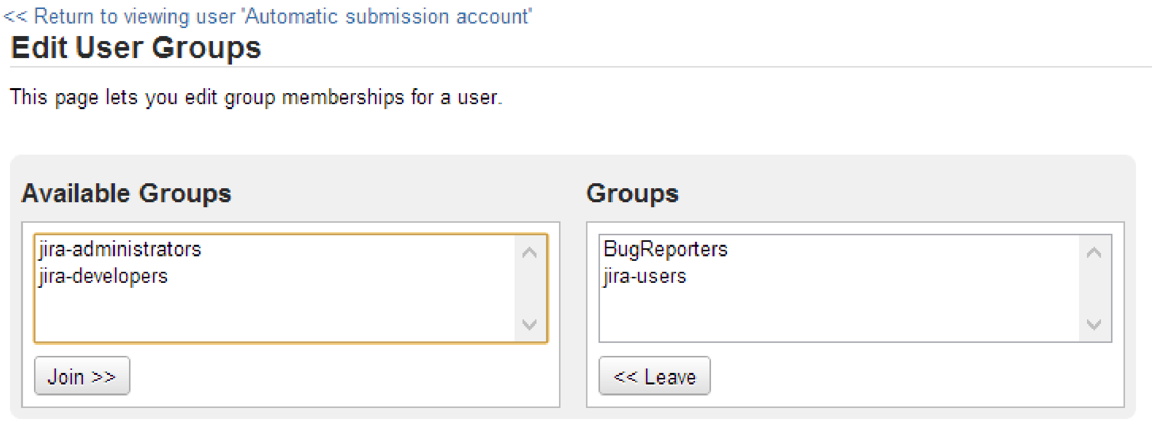
NOTE: When a new project is created, it will be assigned these 'default members' for the 'Bug Reporters' project role. Note that 'default members' apply only when a project is created. Changing the 'default members' for a project role will not affect role membership for existing projects.

- **Return to Project Role Browser**

Default Users	Default Groups
None selected. Edit	BugReporters Edit

Associating role with group

5. Open bug reporting user account and include it into a bug reporting group:



<< Return to viewing user 'Automatic submission account'

Edit User Groups

This page lets you edit group memberships for a user.

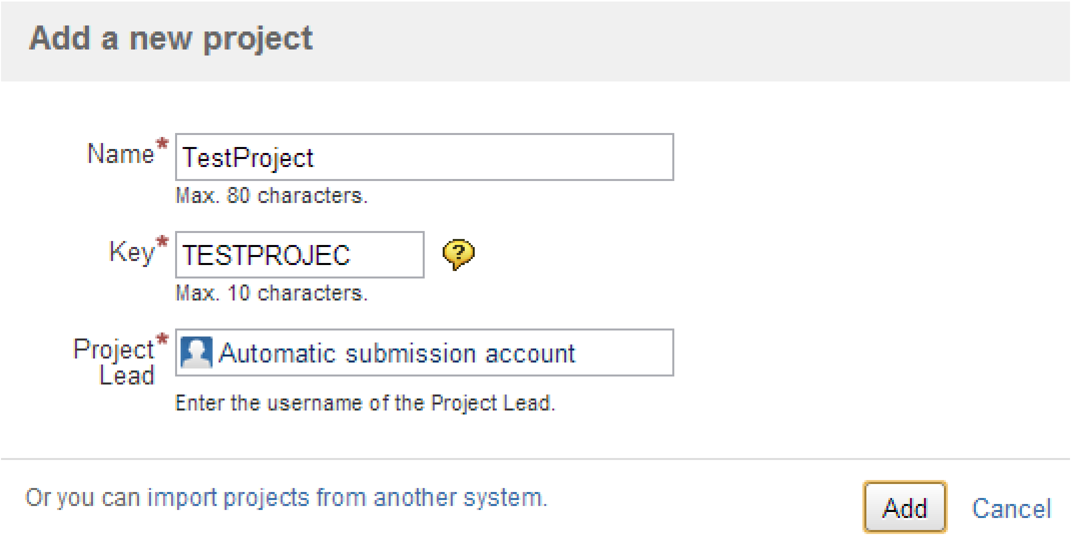
Available Groups	Groups
jira-administrators jira-developers	BugReporters jira-users

Suggested group setup for bug reporting user accounts

Repeat this for all bug reporting accounts.

Creating projects

1. (Administration/Projects) Create project for your software product. You may create several projects - one for each of your software products.



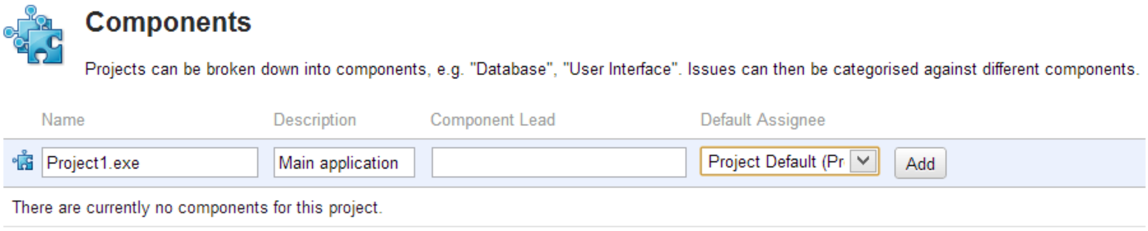
Add a new project

Name*	TestProject Max. 80 characters.
Key*	TESTPROJEC Max. 10 characters.
Project Lead*	Automatic submission account Enter the username of the Project Lead.

Or you can import projects from another system.

Creating new project for bug reports

2. (Optional) Create components for the projects. Typically component is used for identification of the part of your software product.



Components

Projects can be broken down into components, e.g. "Database", "User Interface". Issues can then be categorised against different components.

Name	Description	Component Lead	Default Assignee	
Project1.exe	Main application		Project Default (Pr	Add

There are currently no components for this project.

Creating component(s) for the project

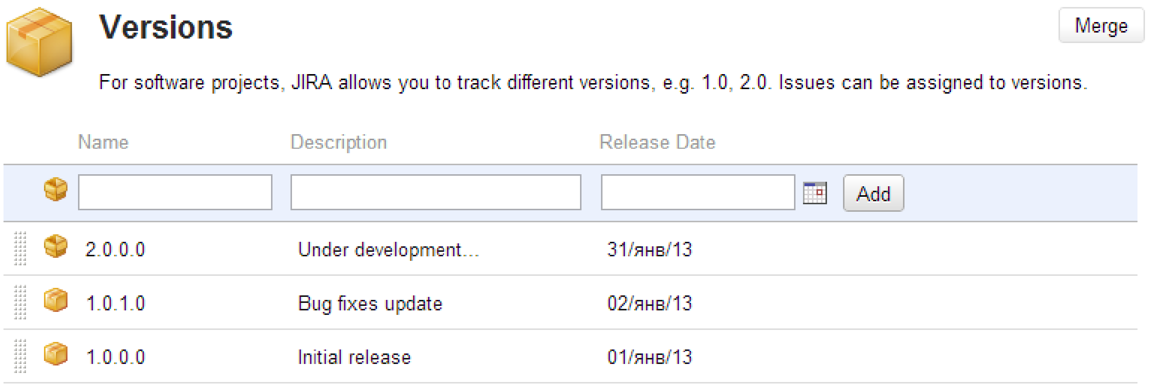
3. Create versions for the project. If you don't use versioning (highly unrecommended) - you can skip this step.

You should create new version for each release of your software. I.e. when you release (publish on site, send to custom, etc) "YourSoftware 1.0.0.0" - you need to create "1.0.0.0" version. When you release update: "YourSoftware 1.0.1.0" - you need to add "1.0.1.0" version.

Version strings can be arbitrary like "1", "1.0", "1.0.1", "1.0.1.0" or even "1.0.1.0 beta 3". However, it's recommended to use four-number versions with optional textual description, for example: "1.0.1.0" and "1.0.1.0 beta 3".

Note: if you don't want to edit project each time you release new version - you can create versions for the future use. I.e. when you release "YourSoftware 1.0.0.0" - you can create "1.0.0.0", "1.0.1.0", "1.0.2.0", "1.0.3.0"..."1.0.10.0" versions.

For example:



Versions

For software projects, JIRA allows you to track different versions, e.g. 1.0, 2.0. Issues can be assigned to versions.

Name	Description	Release Date
2.0.0.0	Under development...	31/янв/13
1.0.1.0	Bug fixes update	02/янв/13
1.0.0.0	Initial release	01/янв/13

Versions

When reporting - version are taken from file's version information, so you must supply the corresponding version in description of your .exe or .dll files.

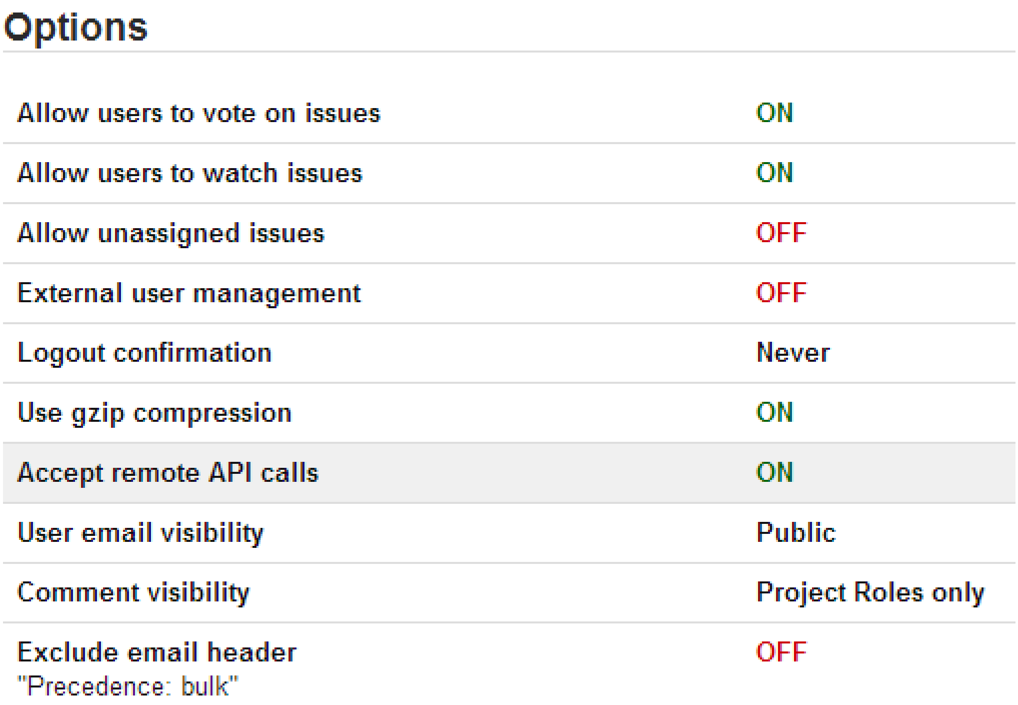
Note: EurekaLog will use closest match. It will also look only for "released" versions.

Important Note: EurekaLog will look only for "Released" versions. Be sure to mark version of your .exe as "Released".

4. Setup project access rights: permissions and roles.

Enabling external access to bug tracker

1. (Administration/General Configuration) Enable access to bug tracker via API calls:



Options

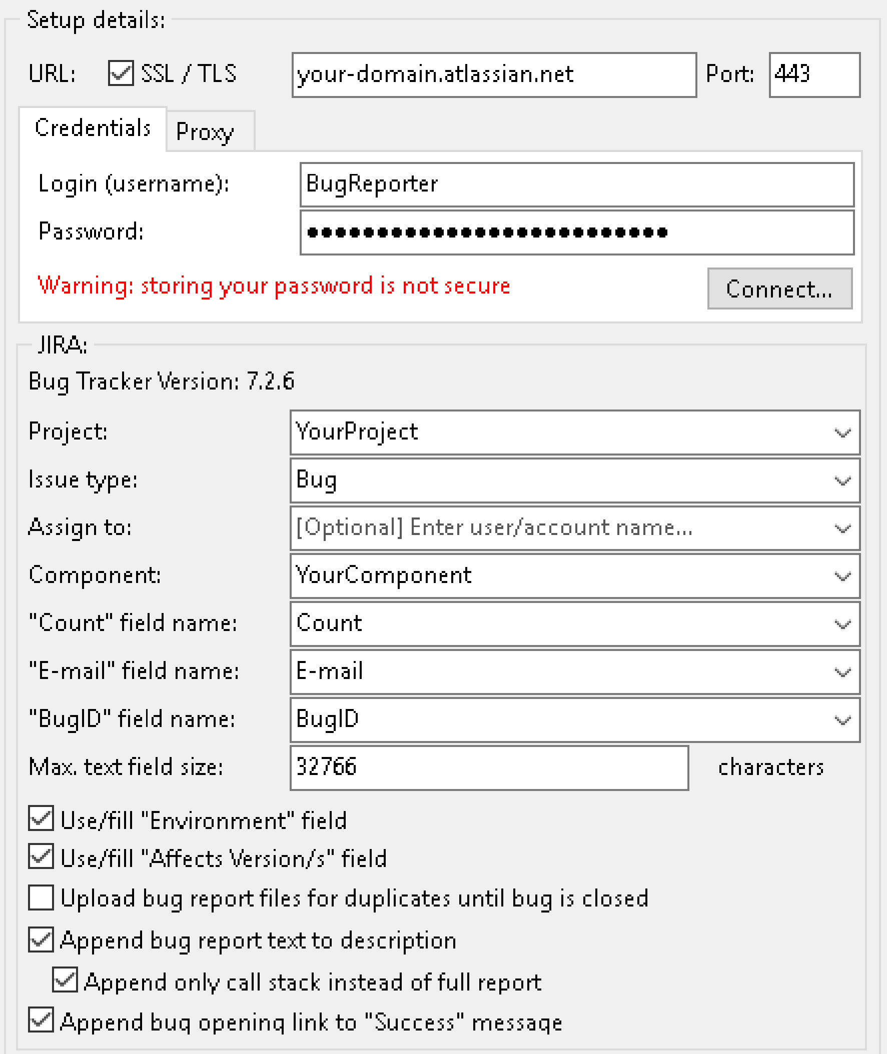
Allow users to vote on issues	ON
Allow users to watch issues	ON
Allow unassigned issues	OFF
External user management	OFF
Logout confirmation	Never
Use gzip compression	ON
Accept remote API calls	ON
User email visibility	Public
Comment visibility	Project Roles only
Exclude email header "Precedence: bulk"	OFF

Enable "Accept remote API calls" option

Note: this option is enabled by default on JIRA cloud accounts. It is also may be unavailable in some JIRA server installations. Please, refer to documentation for your JIRA version for more information.

EurekaLog setup

1. Enter JIRA details into EurekaLog configuration of your projects:



Setup details:

URL: ☑ SSL / TLS your-domain.atlassian.net Port: 443

Login (username): BugReporter

Password: ●●●●●●●●●●●●●●●●●●●●●●●●●●

Warning: storing your password is not secure

JIRA:

Bug Tracker Version: 7.2.6

Project: YourProject

Issue type: Bug

Assign to: [Optional] Enter user/account name...

Component: YourComponent

"Count" field name: Count

"E-mail" field name: E-mail

"BugID" field name: BugID

Max. text field size: 32766 characters

☑ Use/fill "Environment" field

☑ Use/fill "Affects Version/s" field

☐ Upload bug report files for duplicates until bug is closed

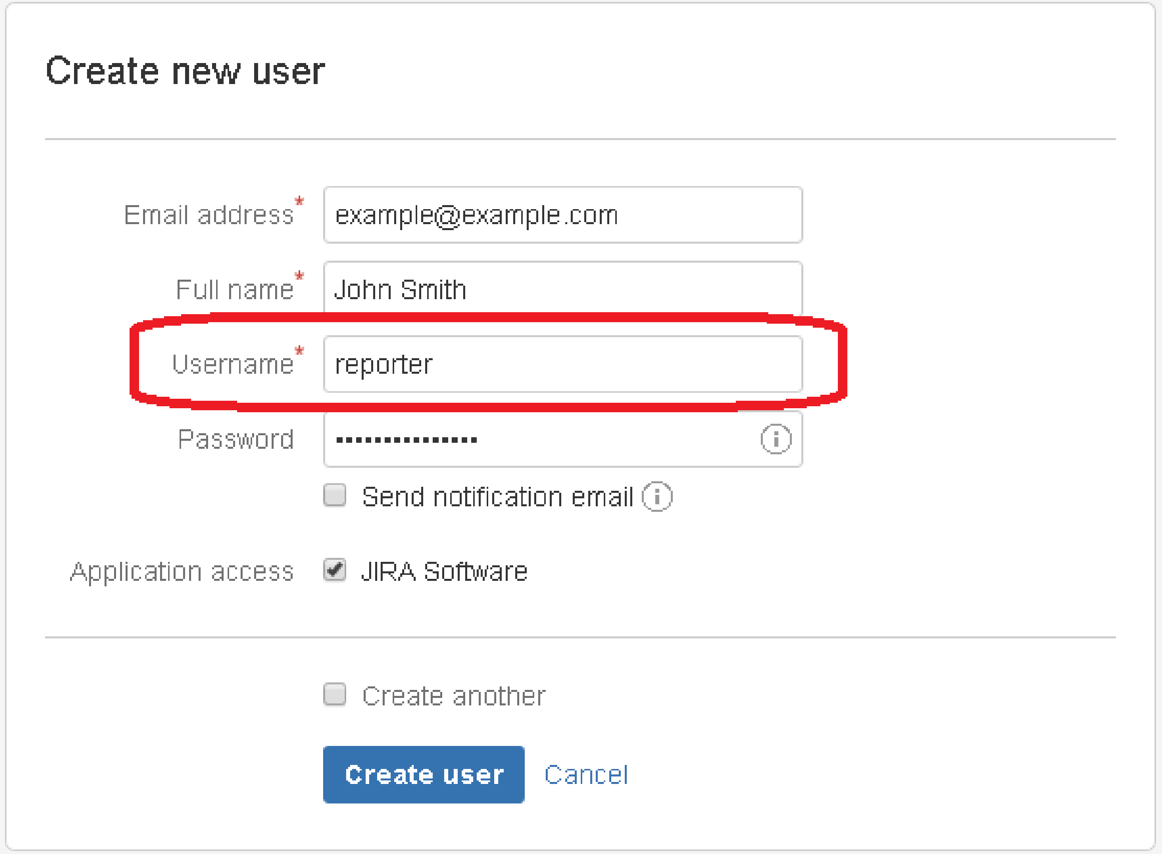
☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

JIRA settings filled into EurekaLog options

Important Note: use your username as login. Do not use e-mail of full name. You are specifying username when creating account for reporting:



Create new user

Email address* example@example.com

Full name* John Smith

Username* reporter

Password •••••••••••••••

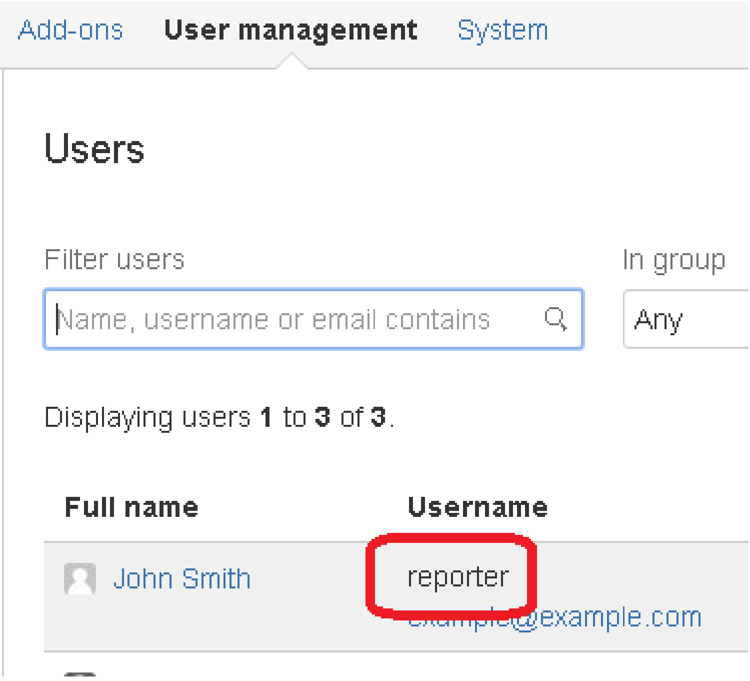
□ Send notification email

Application access ☑ JIRA Software

□ Create another

Username when creating new account

If you forgot your username, you can always find it in your profile or user management:



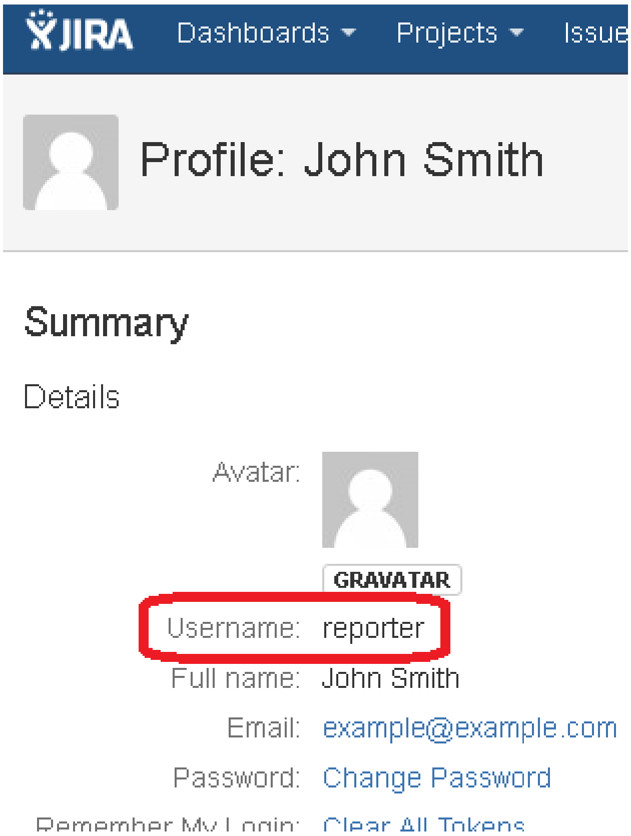
Users

Filter users: Name, username or email contains

In group: Any

Displaying users **1** to **3** of **3**.

Full name	Username
John Smith	reporter example@example.com



Profile: John Smith

Summary

Details

Avatar:

Username: reporter

Full name: John Smith

Email: example@example.com

Password: Change Password

Remember My Login: Clear All Tokens

Locating username of already created accounts

2. Set any additional/common send options 304.

3. Set/fill custom fields. EurekaLog has support for automatic managing of "Count", "BugID" and "E-Mail" fields. You just need to enter field names in EurekaLog options. For other custom fields you have to fill them manually, for example:

```
uses
  EEvents, ESysInfo;

procedure SetCustomFields(const ACustom: Pointer; const ASender: TObject; const AWe
begin
  AWebFields.Values['License'] := GetYourApplicationLicense;
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, SetCustomFields);
end.
```

4. Add any custom data, additional attached files, write necessary event handlers, set exception filters, etc, etc.

Testing

1. Test sending. You can do this right in the EurekaLog send options dialog 302 - by clicking on "Test..." button. This will send test bug report.

Suggested actions are:

1. Click on "Test..." button to test sending and creating of a new bug issue in JIRA.
2. Resolve any found issues (access denied, wrong values in fields, etc).
3. Once successful and there is new issue in JIRA - click on "Test..." button again. This should test updating project.

4. Resolve any found issues (access denied, etc).
5. Once successful - close existing test issue in JIRA (as "Resolved"). Optionally - add a comment with special tags (see customizing feedback 155).
6. Click on "Test..." button again. This should test sending old (already fixed) bugs.
7. Ensure there is no error messages, no problems. You should get "success, this bug is fixed" kind of behaviour. Exact behaviour depends on your settings.
8. Delete test issue in JIRA after testing.

These actions should test that sending is actually working.

2. Now it's time to test your application-specific sending.

1. Place debug code in your application to raise test exception and cause a test leak (if you've enabled leaks collecting).
2. Run your application and invoke this test code.
3. Let application crash and process bug (show dialog, send bug report, etc).
4. Ensure that behaviour is expected.
5. Ensure that you get all files and additional information in JIRA.
6. Remove test code from your application.

Now your application is ready for deployment.

Maintaining projects

1. You need to create or update project versions when you ship new release of your software. If you've created a batch of versions in JIRA for future use - you may skip it until you've run out of versions.

See also:

- Managing bug reports in issue tracker 105
- Security Considerations 158
- Customizing feedback 155

7.2.2 Using unsupported bug tracker software

This article is part of Managing bug report in issue tracker 105 series.

EurekaLog supports these send methods 390. However you may need to send your bug reports into another bug/issue tracking software which is not presented in the list of supported bug trackers. This doesn't mean that you can't use your bug tracker software with EurekaLog at all - you actually can.

E-mail delivery

First method to try is to use e-mail delivery. EurekaLog supports wide range of e-mail sending methods and you can choose one or several methods for delivery bug reports to you. Many bug tracker software includes feature of inserting incoming e-mail as tickets into bug tracker's database - either as integral feature or as 3rd party plugin.

Typically, you need:

1. Create new e-mail account for bug reports collection.
2. Setup your bug tracker software to pick up e-mail from that account (please, refer to documentation of your bug tracker software).
3. Setup e-mail delivery in your EurekaLog-enabled application.

Your bug tracker software may require e-mail messages to be in some predefined format - to avoid inserting spam e-mails into database. In this case you need to appropriately setup e-mail body in EurekaLog's project settings.

Here are examples of e-mail delivery features in bug trackers which are supported by EurekaLog.

- FogBugz
- Mantis (external plugin)

- BugZilla
- JIRA

Surely, usually you don't need to use these interfaces for the mentioned bug trackers - because EurekaLog has support for API of these bug trackers. But these links are examples only. It gives you an idea what to search in the documentation of *your* bug tracker. It serves as starting point for studying.

Search documentation of your bug tracker software for terms "incoming e-mails", "e-mail reporting", "e-mail reports", "e-mail cases", "customer e-mails", etc. You can also ask your bug tracker's support about existence of such feature. You can mention reference to this article, which is available online at http://www.eurekalog.com/help/eurekalog/unsupported_bug_tracker.php

Web-form HTTP posting

Another method to try is HTTP upload method 398. This method is like posting web-form. Some bug trackers allow anonymous bug reporting, which typically includes a web-page which anyone can use to report issues. Usually anonymous reporting is disabled by default (if it exists at all).

You can:
1. Enable anonymous reporting - which makes this web-form accessible.
2. Study this page source to retrieve names of expected fields and possible values.
3. Once you figure out required information:
   - setup HTTP upload method in EurekaLog
   - add OnCustomWebField event handler to supply fields and values as required by web-form

There are two examples available here 115.

Search documentation of your bug tracker software for terms "anonymous reports", "anonymous cases", "anonymous users", "anonymous submitting", "web form", etc. You can also ask your bug tracker's support about existence of such feature. You can mention reference to this article, which is available online at http://www.eurekalog.com/help/eurekalog/unsupported_bug_tracker.php

Note: web-form posting is subject to dependence of UI changes. For example, if you're using web-access method - then you *may* need to use name like "Root project/Your project" to use project created as sub project; category can be specified as "[All Projects] General"; custom field in Mantis will be like "custom_field_1", "custom_field_2", etc.; custom fields in BugZilla have form like "cf_count", "cf_mail", etc. Again, you should look at the page's source to get these names.

Limitations

Surely, work with unsupported bug tracker will be limited.

Here is the list of **possible** limitations:
- no end-user feedback: you may not be able to report back status of bug ticket to end client (like "closed"/"resolved"/"opened") - see also customizing feedback 155
- no file attaches: you may be not able to attach bug report files (i.e. use only text messages)
- no duplication management: each bug report may create new ticket in your bug tracker software
- no user management: reports most likely are anonymous

These limitations are **examples**. It's not necessary that you'll encounter them. It depends on your bug tracker software and its features.

Manual Solution

Finally, there is always an option of manual implementation for your bug tracker. Refer to documentation of your bug tracker to learn about its API. Refer to `ESendAPI`*`xyz`*`.pas` files to learn about reference implementation in EurekaLog. Implement API support in your bug tracker in a standalone unit (follow existing code for supported bug tracker as example). Configure your sending method at run-time.

See also:

- Managing bug reports in issue tracker 105
- Customizing feedback 155
- Customizing EurekaLog 180
- Security Considerations 158

7.2.3 Customizing feedback

This article is part of Managing bug report in issue tracker 105 series.

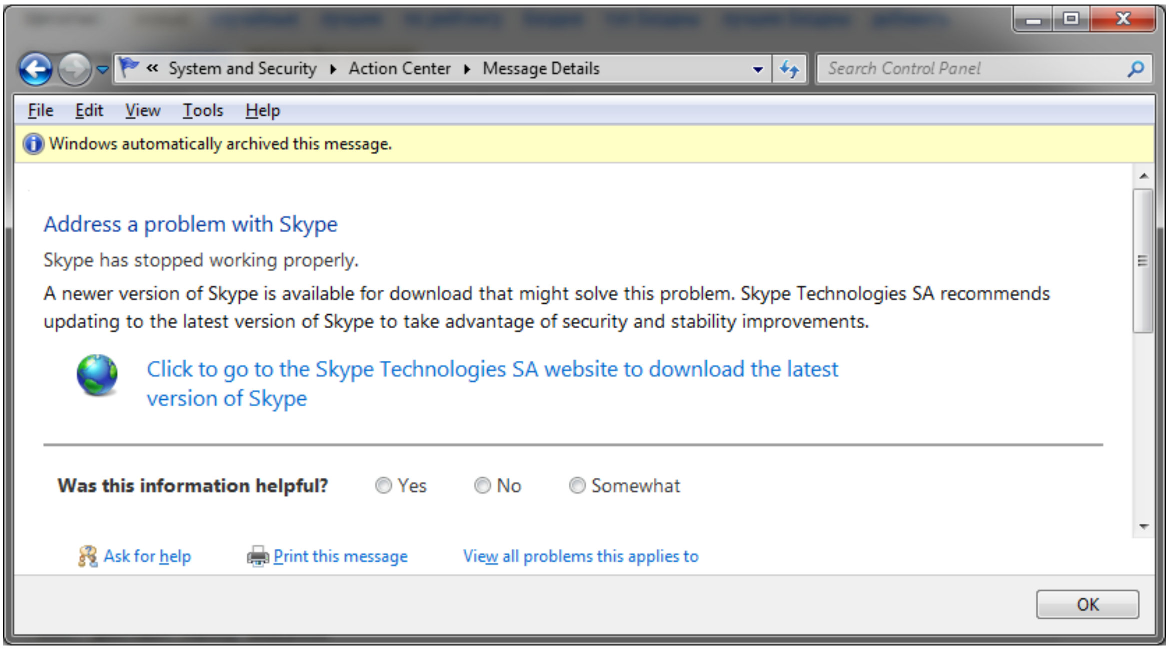
Important note: there are events for customizing sending: such as OnAttachedFilesRequest, OnZippedFilesRequest, and OnCustomWebFieldRequest.

Simple customization includes changing message to fixed static message. For example, message can specify location, where to get patches/updates for your software.

You can do this by changing message in localization options 339.

More advanced approach includes using of dynamic messages and custom HTML-feedback pages. You can include custom message for each bug or open a web-browser with specified page to display extended message or ask user for more information.

To give you feeling of capabilities that you can use - here is an example of application feedback as implemented in Windows Error Reporting 573:


« System and Security ▸ Action Center ▸ Message Details
Search Control Panel
File Edit View Tools Help
Ask for help
Print this message
View all problems this applies to
OK

Windows automatically archived this message.

Address a problem with Skype

Skype has stopped working properly.

A newer version of Skype is available for download that might solve this problem. Skype Technologies SA recommends updating to the latest version of Skype to take advantage of security and stability improvements.

Click to go to the Skype Technologies SA website to download the latest version of Skype

Was this information helpful? ○ Yes ○ No ○ Somewhat

Custom feedback page in Windows Solution Center

You can customize your feedback messages either by using special tags or by using bug tracker features - see below.

Mantis, BugZilla and HTTP upload

Mantis, BugZilla and HTTP upload do not have any special features for customizing feedback. However, you can use special HTML tags.

- **Mantis/BugZilla**: when you're about to close bug - you need to close it with adding message/note. You can write your internal comments (they will not be reported to your users) and you can optionally insert a customization tags.

Note: special tags must be placed in **last** comment to the issue.

- **HTTP upload**: when you've finished processing uploaded files (HTTP form's data) - generate a HTML page with customization tags.

Currently, EurekaLog supports only 2 tags out-of-the-box: **EurekaLogStatus** and **EurekaLogReply**.

EurekaLogStatus tag must contain an integer value, which represents operation status (`TSendResult` type). Most typically used values are `srSent`, `srBugClosed`, `srInvalidInsert` and `srUnknownError`. `srSent` and `srBugClosed` are considered as success status. All other codes are considered as failure. If this tag isn't used, the `srSent` is the default.

Note: it's important to use numeric value (like, '0', '1', '$1'), not name itself (like 'srSent').

You can use this tag to alter status of sending, but usually you omit this tag.

EurekaLogReply tag contains arbitrary string, which will be used as custom message, describing the operation. If this tag isn't used, the message will be default, as usual. If status of the operation is the success (`srSent` or `srBugClosed`), then this message will appear in `SuccessMessage` field of `TResponse` record. If status is failure - message will appear in `ErrorMessage` field. Message will be displayed to user, if you've enabled corresponding options.

Alternatively, you can insert a http:// or https:// link into message. In this case EurekaLog will open a web-browser for this link without showing any other message. Showing HTML page can be used to present "pretty" message, detailed instructions or other advanced messages. For example, if "bug" is not a bug in your software, but problem in run-time configuration, you can insert the URL to your knowledge base article, which describes solution. Another example - you can't solve bug with existing bug report's information. Thus, you close bug and use an URL to web-page, where you ask user to submit more information.

Example of using (example illustrate text of notes which are appending to the issue in bug tracker):

Simple message example

Fixed #304

Inserted additional checks.

SVN commit:
Core.pas
Consts.pas
Design/Designer.pas

<EurekaLogReply>This issue is fixed!
Go to www.example.com to download new version.</EurekaLogReply>

URL example #1

Fixed #304

Not our bug.

<EurekaLogReply>http://www.example.com/kb/issue304.html</
EurekaLogReply>

URL example #2

Fixed #304

```
Can't solve :(  Need more info!

<EurekaLogStatus>0</EurekaLogStatus>
<EurekaLogReply>https://www.example.com/
ask_for_configuration_and_opened_file.html</EurekaLogReply>
```

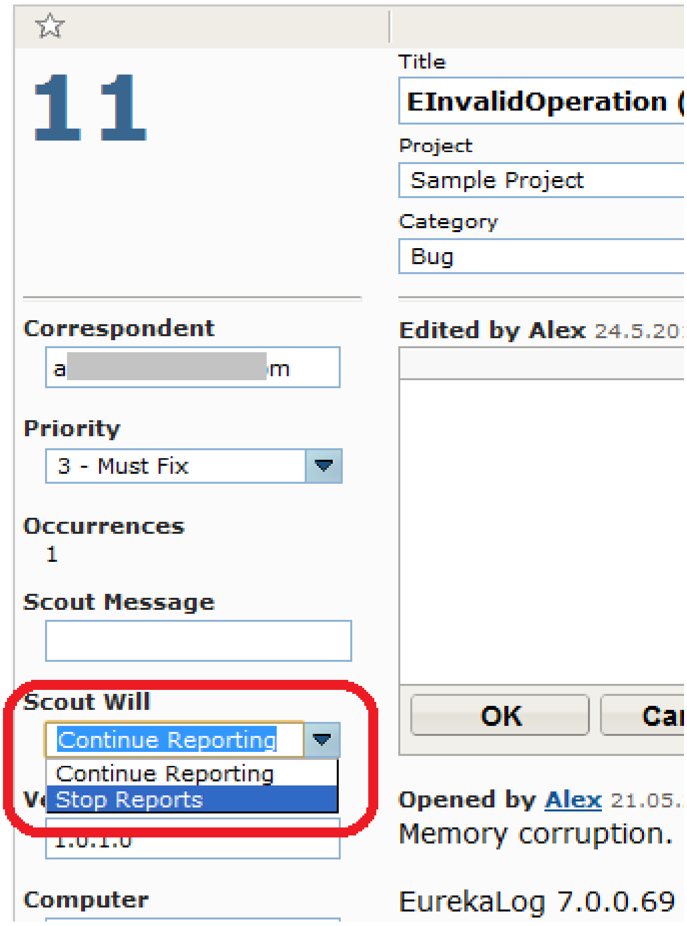
Last example alters send status from default `srBugClosed` to `srSent`.

See also: HTTP upload 398 to get more examples of customizing feedback for HTTP upload method.

FogBugz

FogBugz have feature of controlling field reporting. This feature is used with FogBugz's BugzScout software - a tool to gather report "from the fields". But EurekaLog is perfectly capable of using the same control mechanism.

First, report submission can be stopped at any time by selecting "Stop reports" option:



11

Title: EInvalidOperation (

Project: Sample Project

Category: Bug

Correspondent: a[redacted]m

Priority: 3 - Must Fix

Occurrences: 1

Scout Message:

Scout Will: Continue Reporting

- Continue Reporting
- Stop Reports

Version: 1.0.1.0

Computer

Edited by Alex 24.5.20

Opened by Alex 21.05.

Memory corruption.

EurekaLog 7.0.0.69

Disabling bug report collecting for issue

You can also re-enable report collecting by selecting "Continue reporting. Of course, report collection will be stopped, once issue is closed.

Second, you can use "Scout message" to set a custom message for closed bug reports. This option acts the same as **EurekaLogReply** tag above. You can either put a simple message here or specify an URL to the web-page. You specify developers comments and end-user message separately: use standard note form to create new developer comment and use "Scout message" to set custom end-user feedback message.

See also:

- Managing bug reports in issue tracker 105
- Exception Driven Development 68
- HTTP upload 398
- Customizing EurekaLog 180

7.2.4 Security Considerations

This article is part of Managing bug report in issue tracker 105 series.

1. Storing credentials inside your executable is not secure

EurekaLog requires valid credentials (login/password pair or API token) to send bug reports to your bug tracker software (or as e-mail via SMTP client). This means that login/password/API token must be stored in your executable. While EurekaLog never stores login/password/API token as plain text (passwords are always stored encrypted), but your executable still needs to run on client's machines. This means that your executable is under control of your clients. Someone may use network sniffer to extract password/API token. Or someone may attach a debugger to your executable and wait for EurekaLog to decrypt password/API token before sending bug report. While this is not a common thing - it can be done. **Your password or API token may be retrieved**, even though it is not a simple task.

Important Conclusion: storing passwords inside your executable is not secure!

2. You can avoid storing passwords in your executable

The most bullet-proof solution is just do not store any passwords in your executable. Of course, this would also means that you can not use any send method which requires you to know login/password pair. This leave us with the following send methods:

- Shell (mailto) 391
- Simple MAPI 393
- MAPI 396
- SMTP Server 398
- HTTP (anonymous) 398

Shell and (S)MAPI methods use client's e-mail client to send e-mails. Therefore, EurekaLog will use client's account for sending, no need to provide your own account.

SMTP Server method does not require any credentials at all. The down side is that method is often blocked by firewalls and/or ISPs - as it is a great way to send spam. And even if you manage to send e-mail via SMTP Server method - it will most likely be marked as "spam"/"malware"/"fake" message by recipient's e-mail server. You will have to setup explicit rule to exclude such e-mails from spam filter.

HTTP method is a best choice here - as it does not require any credentials and usually safe with firewalls/ISPs, but it will require you to write code (script) to receive submitted bug reports.

While those are not exactly bug tracker send methods, but you still can use them to post bugs to your bug tracker. For example, you may set up your bug tracker to pick up e-mails from certain address and insert them as reports 153. And HTTP script may use bug tracker's API to insert bugs 115.

In some cases, your bug tracker may provide a ready-to-use HTTP facade. For example, FogBugz provide options for BugzScout and anonymous sending 115, JIRA has Issue Collector.

Important Conclusion: you can hide your bug tracker (e.g. credentials) behind a facade (either e-mail of HTTP web-form).

3. Minimize impact of discovering your passwords

While the above mentioned send methods are resistant against password stealing, but often it is not a convenient way to submit reports. Those methods also have limitations: you may be not able to count duplicate reports 68 and receive feedback 155. An exception to this is HTTP sending, which can act as facade to your bug tracker. You can hide actual login/password for bug tracker in your HTTP receiving script. See these examples 115.

Therefore, most EurekaLog's customers use a compromise way. The idea is that you store password in your executable, but you limit things that you can actually do with this password. Obviously, you should not use your administrator account for sending reports.

Important Conclusion: never use your administrator password in EurekaLog configuration.

What you can do to reduce impact of a leaked password:

Create a separate account for automatic submission of bug reports

You should always create a new account, which you will use with EurekaLog. Limit this account as much as possible. Allow it to read issues, create issues and modify issues (restrict modification to "Count" field only - if such fine access restriction is supported). Limit it to one project only - the one which is used to receive bug reports.

This will prevent attacker from altering administrator settings of your bug tracker, alter other accounts, edit other projects, etc.

The most customizable bug tracker in this aspect is Mantis - as it allows you to create account, which will not be able to modify its own profile. Mantis have very rich access-limiting capabilities. The most non-customizable bug tracker is FogBugz - while it have very nice UI, but it have little customization abilities. Other bug trackers are in between. Please, refer to a specific guide for configuration 107 of your bug tracker to learn more.

Protect your account used for automatic submission when possible

If your bug tracker or send method allows to disable changing account - use this opportunity. For example, Mantis allows you to mark account as "Protected", which means its settings and password can not be modified.

Use API tokens when possible

You can use API tokens instead of passwords (if your bug tracker allow that). While this does not remove threat completely, it will greatly reduce risks - because it is usually not possible to alter settings/password without entering account password (which will not be stored in executable when you are using API tokens). Additionally, you may create and revoke API tokens for single reporter account as you like.

Use two-factor auth when possible

Some systems allow you to use two-factor authentication to log in your account. Use this when possible. This is similar to API tokens. Usually you would need to do two things:

1. Enable two-factor auth for your account. This will block access to attacker, who retrieved password from your application - since attacker would have access to password, but not the second key.
2. Create application-specific password. This will allow EurekaLog to send reports. Application-specific password allows using service, but does not allow logging in.

A good example is so-called "2-step verification" in Google (GMail). Once you enable it for your account - you won't be able to log in to your account from external application, as logging in would require to enter code from SMS sent to your phone (alternatively you can use application or hardware key). However, you can enter your account settings and create so-called "App Password". App Passwords are used to access accounts from applications which do not support two-factor auth. In other words, you can enter App Password to EurekaLog project settings for SMTP Client send method 397 - and EurekaLog will be able to use your account to send e-mails. However, even if attacker extract App Password from your EurekaLog-enabled executable - he would not be able to log in to your GMail account, as this will require to know key from SMS, GAuth application or hardware token.

Create a separate project for automatic submission

You should create a new "project" specifically for receiving bug reports. In other words, you should create two "projects" in your bug tracker for each EurekaLog-enabled application: use one "project" to receive automated reports and use second "project" for manual managing (bugs, new features requests, etc.). You can either move issues between two projects or make references from one project to another.

This will prevent attacker from messing with your sensitive data. Since auto-reporter account will have access only to one "project" (the one which you use to receive bug reports) - the attacker will not be able to view/edit second project (which you use to work manually) or any other project on your bug tracker.

See also:

- Managing bug reports in issue tracker 105
- Detailed walkthough for setup 107
- Using unsupported bug tracker 153
- HTTP upload 398
- Exception Driven Development 68

7.3 EAccessViolation

What is an Access Violation

Every computer program uses memory for running. Memory is consumed by every variable in your program. It can be form, component, object, array, record, string or simple integer. Memory can be allocated automatically for certain types of variables (such as integer or static arrays), the other types require manual control of memory (for example, dynamic arrays). Essentially, from the point of operating system, each variable is characterized by its address (i.e. - location) and size.

Roughly speaking, program uses 3 "types" of memory: area for global variables, the stack and the heap.

Memory for global variables is allocated by OS loader when executable module is loading and it is freed when module is unloading. Global variables are those, which declared outside of class or any routine. The stack is used for allocating memory for local variables (which are declared in some function or procedure) and auxiliary data (such as return addresses or exception handlers). The heap is used for storing dynamic data (such as objects, dynamic arrays, strings, etc.).

Note, that for variables of dynamic types (such as dynamic arrays, strings, objects or components) - though the variable itself is stored in global area or stack, but its data is always allocated on the heap and it (often) require manual control.

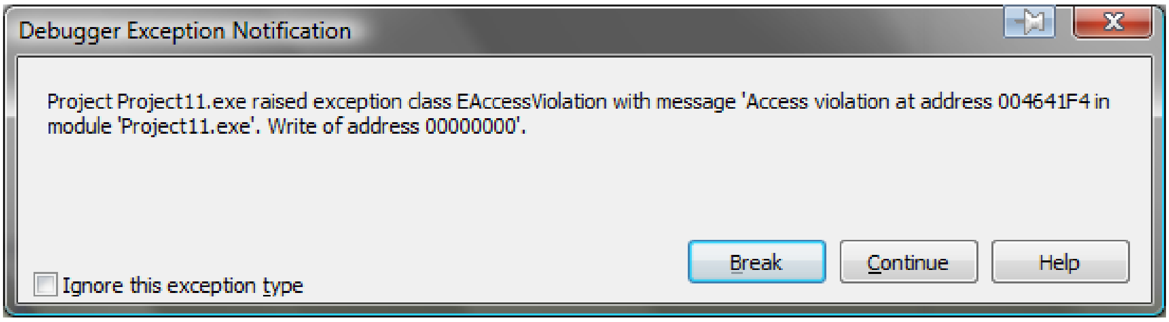
Regardless of who allocates memory for the variable (you manually, or the compiler automatically), memory for each variable must be allocated before its using, and later (when the variable is no longer needed) it should be freed.

Sometimes there can be a situation, where your application trying to get access to certain memory location, which wasn't allocated or was already released - due to bugs in your code. When such things happens - the CPU raises an exception of class EAccessViolation. The usual text for this error is as follows: "Access violation at address XXX in module 'YYY'. Write/read of address ZZZ". Though there is the one simple reason for this kind of error, the real situations for it can be very different.

Looking for source code line of Access Violation

So, what should you do with access violation? Well, first you should try to identificate a source line in your code, where it appears.

If you are getting EAccessViolation while running under debugger:

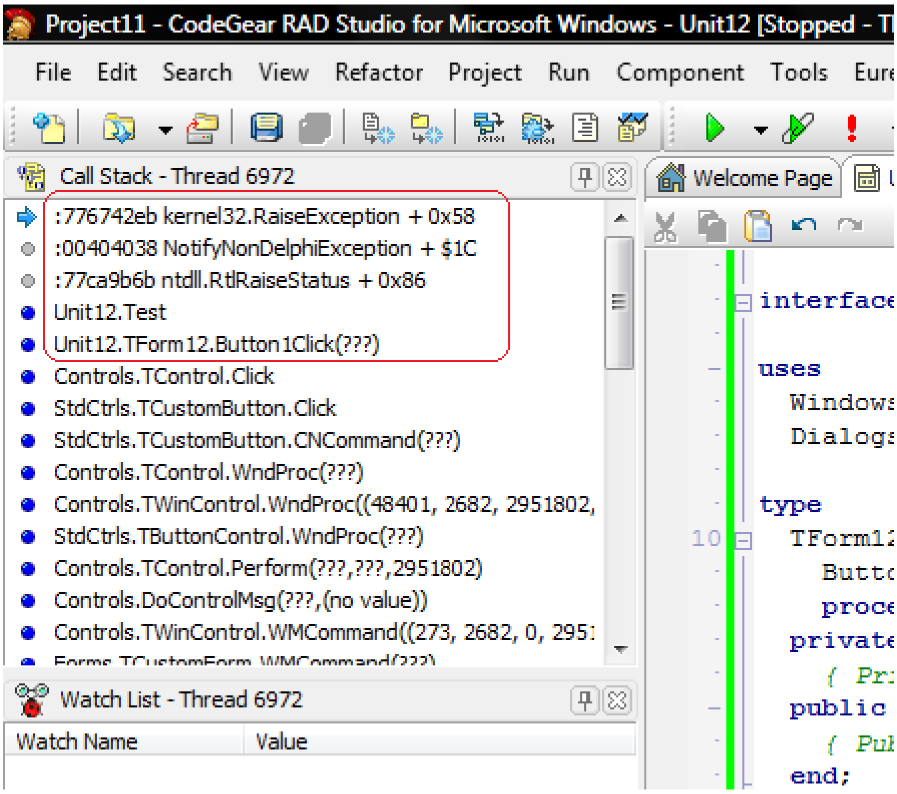


Debugger Exception Notification

Project Project11.exe raised exception class EAccessViolation with message 'Access violation at address 004641F4 in module 'Project11.exe'. Write of address 00000000'.

A typical debugger's notification about access violation exception

Then you should just click "Break" (it is called "Ok" in older Delphi's versions) and the debugger will point you to source line immediately. Additionally you can take a look at call stack by choosing View/Debug Windows/Call stack from Delphi's main menu:



Project11 - CodeGear RAD Studio for Microsoft Windows - Unit12 [Stopped - T

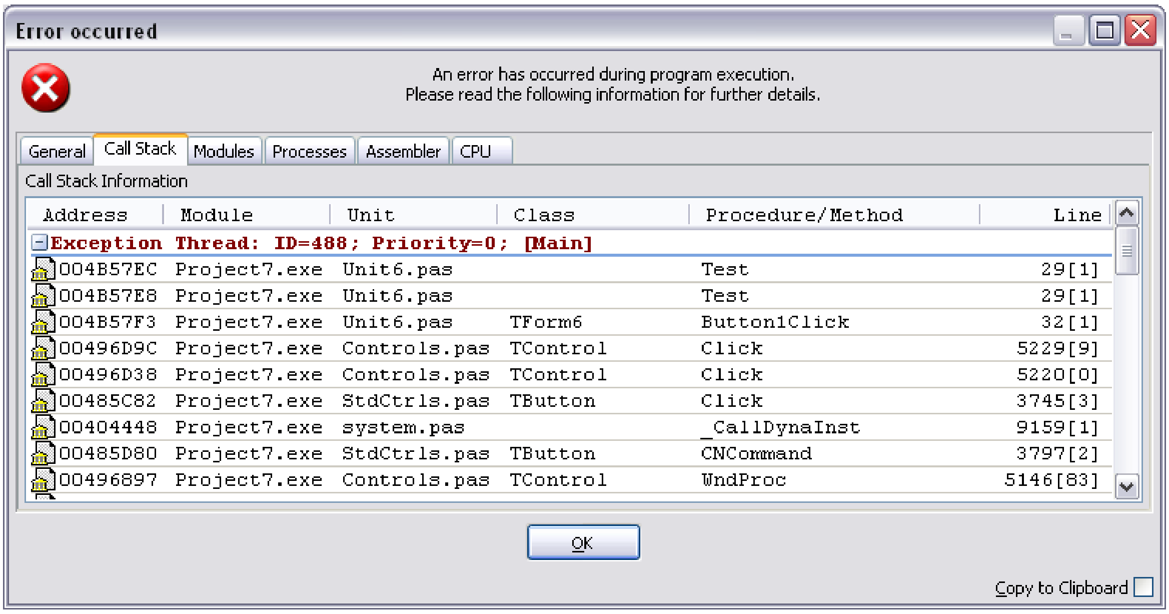
```
:776742eb kernel32.RaiseException + 0x58
:00404038 NotifyNonDelphiException + $1C
:77ca9b6b ntdll.RtlRaiseStatus + 0x86
Unit12.Test
Unit12.TForm12.Button1Click(???)
Controls.TControl.Click
StdCtrls.TCustomButton.Click
StdCtrls.TCustomButton.CNCommand(???)
Controls.TControl.WndProc(???)
Controls.TWinControl.WndProc((48401, 2682, 2951802,
StdCtrls.TButtonControl.WndProc(???)
Controls.TControl.Perform(???,???,2951802)
Controls.DoControlMsg(???,(no value))
Controls.TWinControl.WMCommand((273, 2682, 0, 295
Forms.TCustomForm.WMCommand(???)
```

```
interface

uses
  Windows
  Dialogs

type
  TForm1
    Butto
    proce
  private
    { Pr
  public
    { Pu
  end;
```

A typical call stack as displayed by IDE debugger

This window shows you a call stack - the trace of executing to current code's point. You should read this from top to bottom. The current location is marked by little blue arrow. You can also double-click on line to go to a particular location. For example, if you double-click on "Unit12.Test" line - debugger will show you location where exception was raised.

If you are using an exception tracer tool (such as EurekaLog) then there would be a bug-report instead of usual error message. You can see a call stack in the report (call stack view can differ due to different building algorithm):



An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Exception Thread: ID=488; Priority=0; [Main]					
004B57EC	Project7.exe	Unit6.pas		Test	29[1]
004B57E8	Project7.exe	Unit6.pas		Test	29[1]
004B57F3	Project7.exe	Unit6.pas	TForm6	Button1Click	32[1]
00496D9C	Project7.exe	Controls.pas	TControl	Click	5229[9]
00496D38	Project7.exe	Controls.pas	TControl	Click	5220[0]
00485C82	Project7.exe	StdCtrls.pas	TButton	Click	3745[3]
00404448	Project7.exe	system.pas		_CallDynaInst	9159[1]
00485D80	Project7.exe	StdCtrls.pas	TButton	CNCommand	3797[2]
00496897	Project7.exe	Controls.pas	TControl	WndProc	5146[83]

A typical EurekaLog report about exception

You can see there the same information. And you also can double-click on lines to go to that locations in IDE code editor.

Okay, finding the error's location - this is only half of the case. Determination why there is an error in this line - it is the second half of the case.

Looking for the Access Violation's reason by analyzing the code

If you got an error while using debugger, then it is quite simple - you should place a breakpoint to your problem-line and check all variables and expressions in this line after breakpoint's hit - and here it is, the reason for access violation. Just use the debugger.

If there is only a bug-report - then you should use your telepathic abilities to find out the truth. Those psychic powers are comes with experience and we can help you a little with it - by giving you a list of most common mistakes, which can lead to EAccessViolation exceptions.

1. First, there are all kinds of errors of accessing an array's element outside of its borders. For example, the typical newbie's mistake can look like this:

```
var
  X: Integer;
...
  for X := 1 to Length(List) do // wrong! Should be: for X := 0 to Length(List) - 1
  begin
    // ... do something with List[X]
  end;
```

So, if your problem line contains [] - there is a good reason to validate your expression inside [].

Usually, you should catch errors of this sort at development/testing stage by using "Range Check Errors" option. The point is that such errors are very dangerous, because they may go unnoticed, even more than that - they can destroy the stack, so that you can not get the location of the error. But more on this later.

2. All kinds of messing with arguments. Usually those are untyped parameters and buffer-overflow errors:

```
var
  S1: array of Integer;
  S2: String;
...
  // Wrong:
  Stream.ReadBuffer(S1, 256);     // this corrupts the S1 pointer
  // Correct:
  Stream.ReadBuffer(S1[0], 256);  // this reads data into S1 array

  // Wrong:
  FillChar(S2, Length(S2), 0);            // this damages the S2 pointer
  // Correct:
  FillChar(Pointer(S2)^, Length(S2), 0);  // this clears the S2 string by filling i
```

Usually these errors are catched immediately upon function call. You should just examine a function's documentation to figure out what you did wrong. Check: what function expects to receive and what actually you give to it.

3. Passing data between modules. Well, newbies likes to pass data (especially String) between exe and DLL, without caring much about two different memory managers in modules.

These errors are usually detected at development time.

4. Wrong declaration of functions, which are imported from DLL. The most common mistake is wrong calling convention. If you are getting EAccessViolation just by calling a function from DLL - just carefully verify its declaration. Be sure, that its signature is correct and you didn't forget about `stdcall` or `cdecl`.

Though these errors usually detected at development stage, there can be cases, when wrong declaration will make it at production code.

5. Missing of proper synchronization, when working with threads. If you are using more than one thread in your application, then there can be troubles. For example, you can not access a VCL objects from another thread as VCL is not thread-safe - you should use Synchronize for this. Actually, the problem is encountered when one thread changes the data, which is used by another thread - and that becomes a complete surprise for the second thread.

Unfortunately, the problems with thread are the most complex ones. They are very hard to diagnose. The best you can do is to guarantee, that such things can not happen. If you are in doubt - place you code in synchronize or guard it by critical section, when working with shared variables. Sometimes programmer uses CreateThread instead of BeginThread or TThread and forgets about changing `IsMultiThreaded` variable.

6. Calling a function via invalid procedural variable. For example:

```
var
  Lib1, Lib2: HMODULE;
  Proc: procedure;
...
  Lib1 := LoadLibrary('MyDll.dll');         // one piece of code loads DLL. It can
...
  Lib2 := GetModuleHandle('MyDll.dll');
  Proc := GetProcAddress(Lib2, 'MyProc');   // there is no checks! There can be no
  Proc;                                     // Proc can be = nil -> there will be a
...
  FreeLibrary(Lib1);                        // some code unloads library
...
  Proc;                                     // though Proc <> nil, its code is no l
```

```
                                                // that is why there will be an AV.
```

The whole case is very similar to the next situation.

7. Calling of methods or any other access of objects/components, which wasn't created yet or were already released. You should consider this reason if there is some object variables in your problem line of code. Especially, if you do a manual allocate or free of objects somewhere in your program.

The one part of the problem is that when you destroy an object, its variable is not cleared automatically - it continues to point at invalid memory location. The other part is that local variables are not initialized to zero and contains trash at function's call. The last part: there can be multiple reference to one object/component via different variables. Here are few examples:

```
var
  Str: TStringList;
...
  Str.Add('S'); // Mistake! We forget to create an object by calling Str := TString
...
  Str := TStringList.Create;
  Str.Add('S');
...
  Str.Free; // We destroyed the object, but the Str still points to old location
...
  if Str.Count > 0 then // Mistake! An access to already released object
```

All such memory access errors are dangerous as they may be unnoticed. For example, we can access a deleted object, but our memory manager still wasn't return memory to the system, so our access can be successful.

It's recommended to use FreeAndNil to destroy objects or (better yet) to use interfaces instead of objects - because interfaces are auto-managed types, which will be released automatically.

The situation with local arrays is even worse: the point is that local arrays are allocated in the stack, so there is large areas of available memory at its borders. To make things worse: this memory is heavily used by application (as oppose to the memory, which were released by the object destruction).

For example:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array [0..1] of Integer;
  I: Integer;
begin
  I := 2;             // suppose, that I is somehow calculated in you application
                      // and suppose that there is a bug, and I gets wrong value.
  S[I] := 0;          // this line will damage the return address of Button1Click in
end;                  // there will be EAccessViolation at this line, because the ac

procedure TForm1.Button2Click(Sender: TObject);
var
  S: array [0..1] of Integer;
  I: Integer;
begin
  I := -6;            // suppose, there is another wrong value.
  try
    S[I]     := 1;  // instead of changing an array, we damages an exception handle
    S[I + 1] := 2;
    S[I + 2] := 3;
    Abort;          // there would be a full crash, without any message.
                    // The exception manager detect a damaged stack and will termin
  except
    ShowMessage('Aborted');
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  S: array [0..1] of Integer;
  I: Integer;
begin
  I := -1;            // yes, another invalid value for I
  S[I] := 1;          // we damages the stack again, but there won't be any EAccessVi
end;
```

It is very treacherous situation, isn't it? Depending on how we messed up with the array's index, we can get:
a). Application, which produces the correct results.
b). Application, which produces the wrong results.
c). Application, which raises an exception.
d). Application, which crashes.
To make things worse: the very same application can display any of the above behavior, depending on external conditions, such as OS and Delphi's version, user actions before error and so on.

That is why it is extremely important to use "Range Check Errors" option while you develop and testing your application.

Well, you can also enable it for production code, if you isn't sure that your testing was good enough.

So what exactly should we do with access violation? Well, we have a source line, so we should just look through above mentioned cases and try to apply them to our line of code:

- Do we have the [] in our line? If so: can there be an invalid index here?
- Are there any work with objects? If so: check the logic - is there a too early object's release?
- Do we use a DLL? If so: is a function declaration correct? Does all dynamic data exchanges properly handle?

- and so on.

There can be a great help if we can also use few hints from the data.

Looking for Access Violation's reason by analyzing the data

First, we can retrieve some useful information from error's message itself. Let's remember it:

Access violation at address XXX in module 'YYY'. Write/read of address ZZZ.

Okay, the address XXX points to exact location of code, where exception was raised. This is the same address, which is used by Delphi's debugger and EurekaLog to point you to your line of code. The executable module for this address is also displayed in the error message - as YYY. Usually it is your exe, DLL or some system/third-party DLL. Sometimes, however, there can be cases when XXX do not hold any meaningful value. For example, if there is no YYY in the message of if XXX looks suspicious (less then $400000 or greater than $7FFFFFFF on x86-32), then you definitely have problems either with stack corruption (for example, "c" item from the previous section), of call of invalid function (item 6 or, sometimes, 4 from previous section).

The next useful piece of information is "write" or "read" word. The "write" means that the exception occurred during writing, the "read" means that, well, the problem while reading (quite obvious, isn't it?). That means, that we only need to check write or read parts in the problem source line. For example, if the problem line is "P := W" then we should check P if there was "write" word and check W if there was "read" word in the error's message.

And the last hint comes from ZZZ. Actually, we do not care about exact value, but rather about if it is small or large. "Small values" are something like $00000000, $0000000A or $00000010. The "large values" are, for example, $00563F6A, $705D7800 and so on. So, if ZZZ is small - then your code tried to access an object via **nil** reference. If ZZZ is large - then your code tried to access an object via non-nil invalid pointer. In the first case you should check: why do you try to use nil pointer (or who is the bad guy, who set pointer to nil). In the second case you should search for bad guy, who released the object, but doesn't clear the variable itself.

Apart from error's message, there can be another information, which comes from assembly and CPU tabs in EurekaLog's bug-report.

You can see the assembly listing of your program on the "Assembler" tab. It is provided here only for convenience - that way you do not have to search it somewhere else. This is no additional information there. But on the "CPU" tab - you can see the status of CPU's registers, (part of) the stack and (part of) the memory at the moment of exception raising.

For example, we can look at the assembler listing and see that the problem line involves, say, EAX and EDX registers. We can check that EAX is 0 on CPU tab, which means that we are trying to assign value via nil pointer. Then we take a look at the line of source code, which we learned from the call stack, and we will know the name of the variable. And here's the reason for you: the variable, used in assignment, was = nil.

Of course, to work with this information you need a minimum knowledge of assembler, but it is a quite powerful tool.

7.4 Leaks

While any error in your application is always bad, there are types of errors, which can be not visible in certain environments. For example, memory or resources leaks errors are relatively harmless on client machines and can be deadly on servers.

Memory leaks are a class of bugs where the application fails to release memory when no longer needed. Over time, memory leaks affect the performance of both the particular application as well as the operating system. A large leak might result in unacceptable response times due to excessive paging. Eventually the application as well as other parts of the operating system will experience failures.

Windows will free all memory allocated by the application on process termination, so short-running applications will not affect overall system performance significantly. However, leaks in long-running processes like services or even Explorer plug-ins can greatly impact system reliability and might force the user to reboot Windows in order to make the system usable again.

Applications can allocate memory on their behalf by multiple means. Each type of allocation can result in a leak if not freed after use. Here are some examples of common allocation patterns:

- Allocation via Delphi memory manager wrapper (`GetMem`, `AllocMem`, etc)
- Direct allocations from the operating system via the `VirtualAlloc` function
- Heap memory via the `HeapAlloc` function
- Kernel handles created via Kernel32 APIs such as `CreateFile`, `CreateEvent`, or `CreateThread`, hold kernel memory on behalf of the application
- GDI and USER handles created via User32 and Gdi32 APIs (by default, each process has a quota of 10'000 handles)

Item 1 is called "memory leak" in EurekaLog; items 2-5 are called "resource leak" in EurekaLog.

Why leaks are bad and do I always need to release all memory?

Generally speaking, often mem-leak does not mean any visible problem to a user: application still works. Mem-leaks? So what? Program still do all tasks, that I need from it. This is especially true for client applications: cause they work for a limited amount of time. So mem-leak is not scary - since all memory will be reclaimed at application's exit (refer to Jeffrey Richter's book on native code for more info) and so all leaks will be removed too. No, I don't mean that you don't need to fight mem-leaks here: mem-leak is always bad. It is just that mem-leaks aren't that fatal. Of course, this is not applicable for server applications. Server applications work for long period of time, so even minor leak will be deadly.

Some other question, which is close related to above, is: "if all memory is reclaimed upon app's shutdown - can I skip cleanup for global variables? They still will be deleted by automatic cleanup from OS!"

Well, the formal answer is: "you can do it". This is correct and you really can do it. But "can" does not mean "should". Obviously, there will be no technical problems with that approach. So, why is this bad?

Because you can't find a real mem-leaks, if you do like this. If you don't pedantically clean all of your resources - you will get a bunch of mem-leak reports. Well, leaks "by design": technically it's a leak (since resource wasn't released), but it is not a logical leak. Since you know, that those aren't reports about real mem-leaks - you will ignore them. And the problem is that if there will be a report about real leak - you may just miss it.

That's why it is a common "good rule" to always clean your resources. Unfortunately, there can be cases, when you can't do that. Those are very rare cases, but it can happen. But general rule is: always clean your resources, if you can do it. Don't rely on system's cleanup to throw out garbage for you. This will greatly simplify your life in the future.

Wrong approach

When newbie is inspired of the idea of catching memory leaks - he usually opens the Task Manager and watches "Mem usage" column.

So far so good, but then he suddenly notices that this column behaves very strange, even in a simple application: the memory is not decreasing when closing forms, but decreases at minimizing application, etc, etc. A good question: why does newbie think that this column represents memory allocated by his code? If you open "View"/"Select columns" menu - you'll see many other counters, which also matches "memory definition".

So, I'll open a little secret here: the "Mem Usage" column in Task Manager represents amount of RAM, occupied by your application. This value is not the memory, allocated by your code (you can figure out this by yourself, when you first encounter disappearing of memory at minimizing - of course, no one is going to free your memory without your permission). Your application can use less RAM, then your code allocates, since it can be swapped out to page file. And besides, RAM is spend for code too - namely, for system libraries. System libraries are loaded in every process, but there is only one copy of each DLL in system's memory! (I mean only code here). This value is also called "Working Set". You can see many memory-related values in Task Manager by configuring columns. Or you can use Process Explorer tool (add more columns into process list view too).

So let leave our Task Manager for a while and take a look at Pascal. How Delphi manages its memory? All memory in Delphi application is controlled by memory manager. You can change the memory manager in your application by calling SetMemoryManager. That means that you can detect memory leaks very easy - by installing analyzer stub for memory manager.

What does it mean that your application has a memory leak? Well, this means that your code allocates some memory (object, string, array, etc) and forgets to release it. Forgetting about memory's block means that this memory will still be there at application's exit.

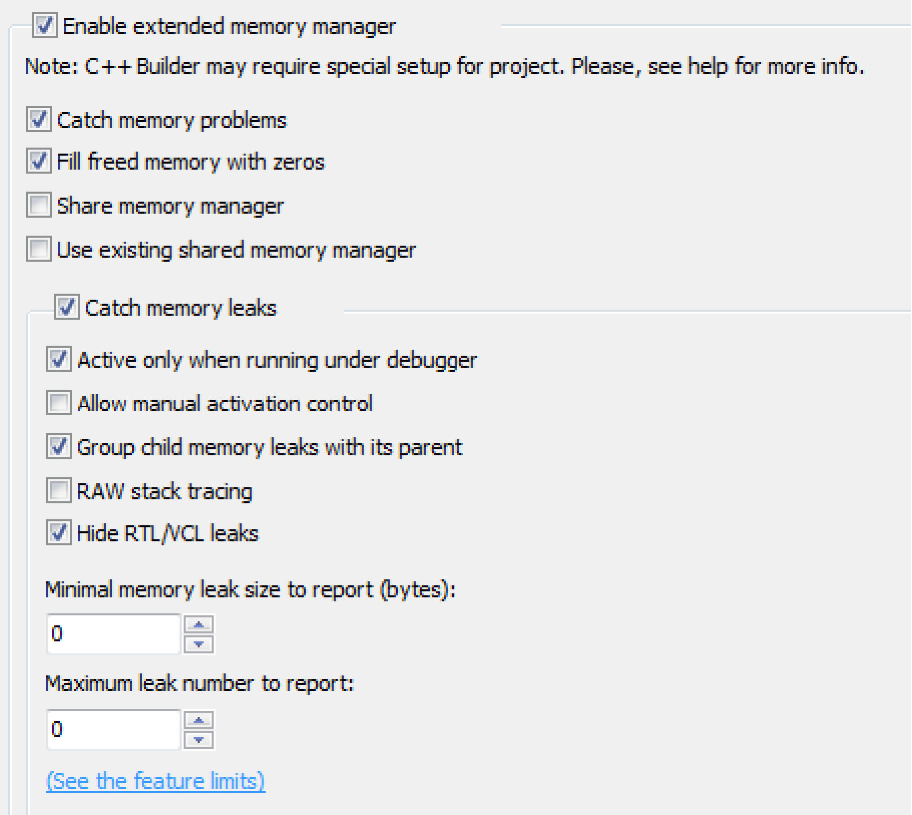
So, to catch all memory leaks you need to enumerate all busy memory blocks at application's exit. Every such block will represent a memory leak.

Using EurekaLog to find memory leaks

EurekaLog has a functionality of catching memory leaks too. It is off by default - because it is not free for your application. Enabling this functionality means a little slow down and increased memory usage[589]. This feature has its limits[589], but it can be very useful for debugging memory leaks on client's machines.

Note: memory leaks catching feature in EurekaLog is made as light-weight - to minimize performance/resource impact on your application. Thus, you can use it in your application deployed on end-user machines. However this means that EurekaLog provides less information than debugging solutions. The primary target of EurekaLog is to let you know about the problems. Surely, you can use EurekaLog to debug problems too, but it can be not suffice in some cases. So you may need to use other debugging tool which is designed to debug problems, not to report them "from the fields".

In any case, to enable this feature - you need to check "Catch memory leaks" option on "Memory problems" tab[250]:



☑ Enable extended memory manager

Note: C++ Builder may require special setup for project. Please, see help for more info.

☑ Catch memory problems

☑ Fill freed memory with zeros

□ Share memory manager

□ Use existing shared memory manager

☑ Catch memory leaks

☑ Active only when running under debugger

□ Allow manual activation control

☑ Group child memory leaks with its parent

□ RAW stack tracing

☑ Hide RTL/VCL leaks

Minimal memory leak size to report (bytes): 0

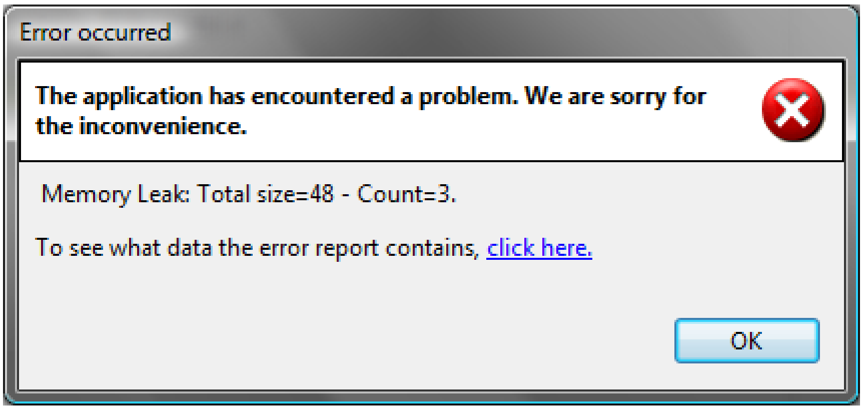
Maximum leak number to report: 0

(See the feature limits)

Memory debugging options

There are bunch of options which controls memory leaks checks activation and its behaviour. See this section 250 for more details about each option.

Anyway, if you activate memory leaks checks and there will be a memory leak in your application at run-time - there will be a usual error dialog at application's exit:



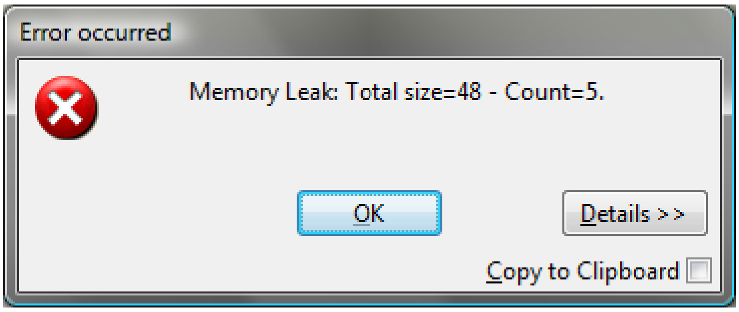
Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Memory Leak: Total size=48 - Count=3.

To see what data the error report contains, click here.

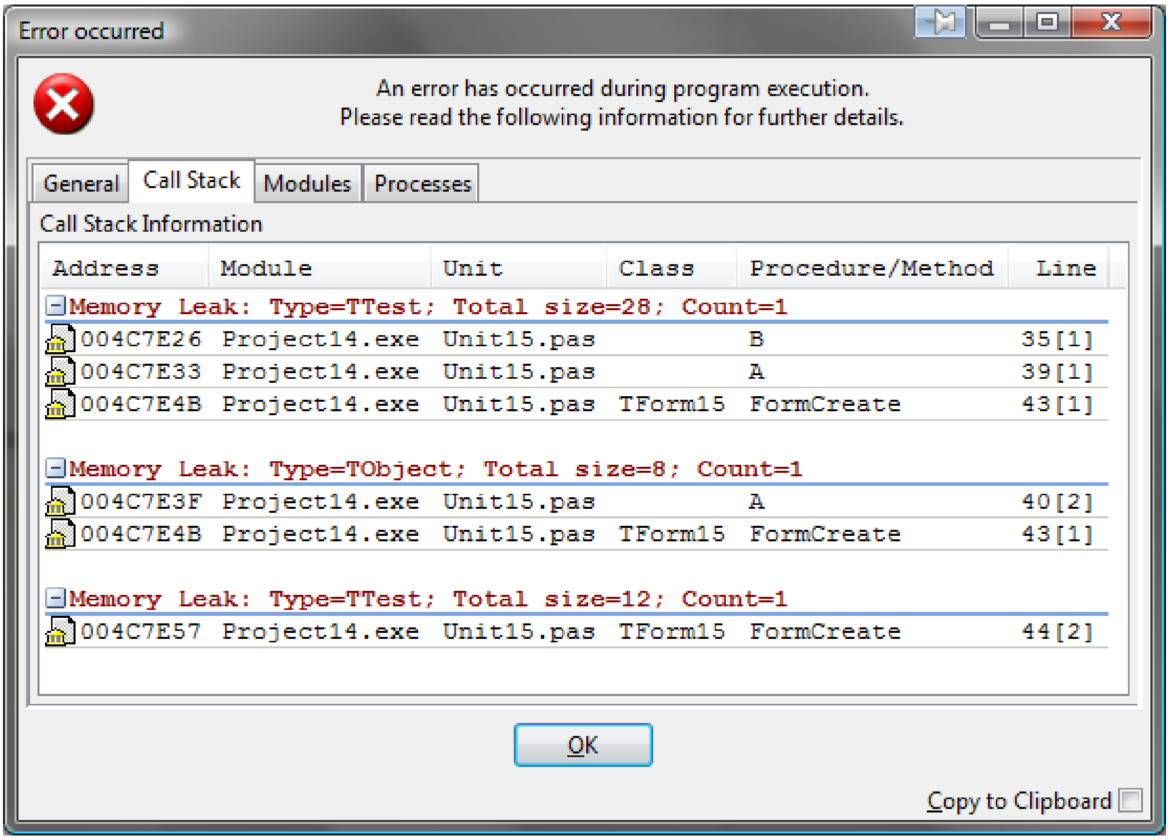
Memory leaks in MS Classic style dialog



Error occurred

Memory Leak: Total size=48 - Count=5.

Memory leaks in EurekaLog style dialog



Error occurred

An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Memory Leak: Type=TTest; Total size=28; Count=1					
004C7E26	Project14.exe	Unit15.pas		B	35[1]
004C7E33	Project14.exe	Unit15.pas		A	39[1]
004C7E4B	Project14.exe	Unit15.pas	TForm15	FormCreate	43[1]
Memory Leak: Type=TObject; Total size=8; Count=1					
004C7E3F	Project14.exe	Unit15.pas		A	40[2]
004C7E4B	Project14.exe	Unit15.pas	TForm15	FormCreate	43[1]
Memory Leak: Type=TTest; Total size=12; Count=1					
004C7E57	Project14.exe	Unit15.pas	TForm15	FormCreate	44[2]

Memory leaks in detailed dialog

As you can see: all memory leaks will be gathered in one single report, which can be send to you as any other EurekaLog report. The only differences from other kinds of reports are: no CPU and Assembler tabs and no calling of event handlers.

How to resolve memory leaks

Many people seems to miss the whole point of mem-leak reports. Typical approach of working with bug report for many people is to open code location from call stack and analyzing it. The problem is: **memory leak report does not point to the problem**.

Note: be sure to read Reading and understanding bug report 72 first, especially Searching bug's location 97 part.

Let's think for a second: what is a leak? Leak is... well, it is when we allocate something and do not release/free it. So, mem-leak report can (and, actually, will) contain that "something" - a resource; and it contains "allocation" - i.e. call stack to line of code, which allocates resource. But where is our problem? An actual problem is sitting at "release/free" moment! A tool can not know: where did you (your code) planned to release resource. That's why report contain only information about allocation. There is no direct information on the problem in the report.

What does it mean, "the problem is in release"? It means, that either we **lost pointer to resource** or we do have a pointer, but our **release routine wasn't called** for some reason. And those are points, which you should look at.

So, what should you do with mem-leak report? Well, you first need to follow call stack and find code. But the next thing is different (comparing to exception bug report): you don't need to analyze **this** line. You need:
1. Note, what resource was allocated here (object, string, array, memory block, etc...).
2. Find, where this resource should be released "by plan" (call do destructor, out of scope, explicit free call, etc...).
3. Found reason, why resource wasn't release at founded location.

As it was already said: there can be 2 reasons for item 3 - either we lost reference or we missed the call.

Delphi's bugs

Before starting doing anything - make sure, that the problem really exists: run your application in wild run without debugger. This will eliminate any possible false mem-leaks like this.

Aside from IDE's bugs, there can be bugs in RTL/VCL too: example. It can be direct bugs (and there is change for their fix in next Delphi version - example), or things that just can't be fixed. Anyway, both cases introduce a mem-leaks in your application and your code has nothing to do with it.

So what can you do here? Putting patching apart - the only thing you can do is to ignore them (since you can't fix them). Yes, this is a workaround. You don't fix a problem - you just hide it, so you can concentrate on problems, which you can fix. The main danger here is overuse of such routines: do NOT add all mem-leaks as "registered" - don't forget that this will not fix the problem!

Best Practices

Certain coding and design practices can limit the number of leaks in your code:
- Use managed data types with reference counting and smart pointers wrappers for non-managed types and functions (you will need to write your own wrapper classes).
- Be aware of leak patterns with managed types: circular references between objects.
- Avoid using multiple exit paths from a function. Allocations assigned to variables at function scope should be freed in one particular block at the end of the function.
- Use try/finally blocks to ensure (guarantee) finalization and dispose of memory and resources.
- Be careful with type-less functions in RTL. Any code which works with untyped argument must be carefully analyzed for leaks possibility.

See also:
- Configuring project for leaks detection 508
- Other memory problems 171
- Memory leaks settings 250
- Resource leaks settings 255
- EurekaLog memory leaks detection limitations 589
- EurekaLog resource leaks detection limitations 589

7.5 Memory problems

How to diagnose and fix memory problem

If you have memory corruption issue and you got a report for it - this report will be a simple indication that you have a problem. You **won't be** able to fix the problem by using this report. Why? Because any such report is a note that the problem **had** occurred *somewhere* and *some time* ago. It's somehow similar to memory leaks - and we've already discussed it earlier 166. The problem is that nobody can scan each CPU instruction and ask: "is this command going to corrupt my memory?" That's why all checks are performed from time to time at certain checkpoints. Besides, only special data can be validated automatically. For example, if we take mem-leaks case - the checkpoints are calls of memory manager's routines and verified data are internal structures and freed memory. But even in that simplest case memory manager does not scan the entire memory pool on each request, limiting check to one memory block in question only. This is a usual trade-off between speed and functionality.

Okay, so, having a report, you will know that there is a problem. But you don't know where is it. You have a chance to locate it in the case of memory leaks, but not in the case of memory corruption. That's because you have some references to code for leaks, but references to code for memory corruptions are total off-topic. The real culprit-code can sit a million instructions away in space and time from the code, which crashed because of it, and there is no references to it. That's why the very first thing, that you should try to do (wherever you have a report or just crash/hang) is to try to **reproduce** the problem. Sometimes you can do it easily; sometimes it is possible, but hard to do; and often it is just not possible at all.

If you've managed to reproduce the problem - then it is a very simple case. Just debug your application as much as you want. The most useful tools here will be memory breakpoints. General strategy is simple: you need to find a moment, when memory is committed, but is not corrupted yet. You place a break-point on the memory (yes, Delphi's debugger can do it; we'll not discuss it here - please, refer to other resources or Delphi's help) and you just run your application. As soon as this break-point fire - you'll find the culprit for memory corruption. Make yourself at home and take your time: analyze the call stack, variables, etc, etc - the situation is under your control.

So, to put it short: the main question here is to locate the problem (assuming you can reproduce it at all). We'll discuss the different methods below, which you can use to locate the problem. Some of them you can use always - both in debug and release version. Some of them are only applicable to debug version.

If you aren't able to solve the problem (either you can't reproduce it or you can reproduce, but can't locate it) – then the only options is to use passive methods. I.e. things, which aren't directed to your **particular** issue, but rather helps you to **improve** your code - that way after improvements you'll be able to diagnose the problem or it may be that the problem will go away without doing anything specific. For example, if your code is chaotic mix of totally unrelated routines calls without slightest sign of logic - you can spend half a year looking for the reason (and still not solve it). Or you can spend few months to refactor/improve your code - and then hunt down and fix not only this problem, but other issues too, which you've spotted because your code becomes much clearer.

Problem's locating (active methods)

First at all, you should analyze, what can be your problem. There are two main cases here: **dynamic memory (heap)** or **the stack**. Depending on the answer you may use methods for the heap or for the stack. For example, using debugging memory manager can help you with the memory corruptions in the heap, but it can do nothing about stack corruptions. If you aren't sure about it - just use all methods.

1. Using debugging memory manager (heap). Debugging memory manager is any memory manager, which provides additional features for debugging memory problems. Searching for memory leaks and searching for memory corruption bugs use the very similar approach. EurekaLog's case: these checks are enabled on memory problems page 250. Other options may affect the results too, but these ones are primary options for memory corruption checks. Just enable additional options and run your application, until debug memory manager will catch a problem.

Please note that EurekaLog uses light-weight methods, which are fast and can be used on end-user machines. However, these methods may be not enough for you to local debugging. In this case you should use specialized heavy-debugging code - like FastMM, SafeMM, AQTime, etc. This will show down your application a lot and you can not use it on end-users machines, but you can stress-test your application locally, at developer's machine.

2. Enabling debugging options (stack and heap). We mentioned this 225 before too. The main option here is "Range check errors", which allows you to catch out of range errors in array-based data structures (note, that this option have a bug in old Delphi's versions). Besides this option, you may want to disable inlining and optimization (to simplify debugging and to avoid bugs like this). Unfortunately, Delphi's compiler do not have a more generic option for checking stack's state like others compilers have.

3. Forced checkpoints (stack and heap). As already mentioned, any report about memory problem reports only about moment of detection, not about the problem itself. You must locate the problem. But how can you do it? Obviously, you need to find a point, when problem is not occurred yet (memory is not corrupted); find the point, when memory is corrupted. Therefore, the problem will sit somewhere between those two points. Each of these moments will be a **checkpoint**. By moving (or creating) checkpoints - you can reduce code's area with problem until you locate it. Sometimes, those checkpoints are created automatically. For example, debugging memory manager validates memory block each time its routine is called for this block. For the stack: it can be routine leave. Since you successfully leave the routine - this means that return address wasn't damaged, so there was no stack corruption (at least some type of it). If those automatically created checkpoints aren't suffice to locate the problem (or they aren't created at all) - then you need to create them manually.

We have an option to force check manually for the heap. You can call CheckHeap routine for EurekaLog. You force memory manager to scan the entire memory pool for corruptions by calling this routine (obviously, only consistence of internal info/headers can be validated, not the data inside memory blocks). By putting calls to this function around the code - you put explicit checkpoints. Start with calling them periodically. Once you found a problem between two calls - move them closer to each other, until you locate the problem.

You can also switch to SafeMM for even more debug control.

4. Debug checks (stack and heap). It's not always possible to use or set checkpoints as discussed in previous item. For example, no one can check consistency of **your** information, all automated tools can check only their info, not yours. That's why you may need to validate your info manually. Well, it's simple: just place as many checks as you want around your code. Put Assert's call everywhere. Check every thing, that you're able to check. Once you found a problem between two Assert's call - move them closer to each other, just like in checkpoint's case. As soon as you reduce a gap enough to acquire the address of corrupted memory - you're done. Just run your application until the moment before problem and put a memory-breakpoint on this address (see also below).

5. Avoiding local variables (stack). Since we don't have much tools for the stack - you can move the problem elsewhere by avoiding local variables: try to use global variables (just for test, of course) or (better yet) put all local variables into record, which you allocate dynamically in the heap. This will move the problem to another area, where we have some handy tools (your favorite debugging memory manager).

6. Problem with threads (heap). Multi-threading usually does not affect stacks, but it can be a reason for many hard-to-detect problems with global or heap's data (well, not multi-threading by itself, but rather synchronization errors). Debugging of multi-threaded application is large and complex thing, so it won't be discussed there - please, see other resources.

7. Memory breakpoints (stack and heap). If you'll found an specific address for memory, which was corrupted, things will become much easier. All you need to do now is to use memory breakpoints. Memory breakpoints is handy ability of Delphi's debugger, which allows you to put breakpoint on memory, just like you do this for code. A memory breakpoint triggers, when some code accesses memory. Use Delphi's help to learn details on how to use them.

So, you have a memory's address. Run your application until the moment, when this memory will be available (allocated). It should be in the valid state at this moment. Place a memory breakpoint on it. And run your application. When breakpoint fires - check the code, which caused it. You'll find the culprit eventually.

So, if you wasn't able to solve your problem with the above methods - then the only thing left is:

Prevention of problems with memory (passive methods)

1. Avoid low-level code. It's simple: scan all your code, looking for calls of low-level routines (which aren't type-safe, therefore have a high chance of corrupting memory). Double-check all usage cases. Replace low-level code with high-level counterpart, if you can do it. It's better to do it slow and safe/correct than do it fast, but incorrect.

2. Check code for being unicode-ready. Most common error is confusing length and size - i.e. size of buffer in characters and size of buffer in bytes.

3. Use wrappers. Separate all API code into separate unit/class, which you can validate as single entity. You'll reduce searching area and simplify code by placing suspicious/potential troublesome code in the same place.

4. Code review by other developer. It's well-know fact, that your eyes see only things, which you brains want to see. That's why it's good thing to give your code to colleague - sometimes he/she can spot obvious problem, which you can't solve for few hours/days.

Actually, this section is endless. There are many books, which tells you how to write a good quality code. And they do this in more details, than we can do it here. That's why we won't list anything further - just give you some advice: read "smart" books. Consider the text above only as short example. You can improve yourself and your code by reading books and blogs. Many problems will be easier to spot or they can disappear eventually.

7.6 Hangs and deadlocks

Hangs - User Perspective

Users like responsive applications. When they click a menu, they want the application to react instantly, even if it is currently printing their work. When they save a lengthy document in their favorite word processor, they want to continue typing while the disk is still spinning. Users get impatient rather quickly when the application does not react in a timely fashion to their input.

A programmer might recognize many legitimate reasons for an application not to instantly respond to user input. The application might be busy recalculating some data, or simply waiting for its disk I/O to complete. However, from user research, we know that users get annoyed and frustrated after just a couple of seconds of unresponsiveness. After 5 seconds, they will try to terminate a hung application. Next to crashes, application hangs are the most common source of user disruption when working with GUI applications.

There are many different root causes for application hangs, and not all of them manifest themselves in an unresponsive UI. However, an unresponsive UI is one of the most common hang experiences, and this scenario currently receives the most operating system support for both detection as well as recovery. Windows automatically detects, collects debug information, and optionally terminates or restarts hung applications. Otherwise, the user might have to restart the machine in order to recover a hung application.

Hangs - Operating System Perspective

When an application (or more accurately, a thread) creates a window on the desktop, it enters into an implicit contract with the Desktop Window Manager (DWM) to process window messages in a timely fashion. The DWM posts messages (keyboard/mouse input and messages from other windows, as well as itself) into the thread-specific message queue. The thread retrieves and dispatches those messages via its message queue. If the thread does not service the queue by calling `GetMessage`, messages are not processed, and the window hangs: it can neither redraw nor can it accept input from the user. The operating system detects this state by attaching a timer to pending messages in the message queue. If a message has not been retrieved within 5 seconds, the DWM declares the window to be hung. You can query this particular window state via the `IsHungAppWindow` API.

Detection is only the first step. At this point, the user still cannot even terminate the application - clicking the X (Close) button would result in a WM_CLOSE message, which would be stuck in the message queue just like any other message. The Desktop Window Manager assists by seamlessly hiding and then replacing the hung window with a 'ghost' copy displaying a bitmap of the original window's previous client area (and adding "Not Responding" to the title bar). As long as the original window's thread does not retrieve messages, the DWM manages both windows simultaneously, but allows the user to interact only with the ghost copy. Using this ghost window, the user can only move, minimize, and - most importantly - close the unresponsive application, but not change its internal state.

The Desktop Window Manager does one last thing; it integrates with Windows Error Reporting, allowing the user to not only close and optionally restart the application, but also send valuable debugging data back to Microsoft. You can get this hang data for your own applications by signing up at the Winqual website.

See also: WER 573.

Hangs - EurekaLog Perspective

EurekaLog's hang detection works similarly to system's one. If you enable hang detection 257 - then EurekaLog will create a new thread on startup of your application. This "hang detection" thread will constantly ask UI thread to process a `WM_NULL` message - this is the message that do nothing. So it can be used for window polling. If an application window is hung, it will not be able to process the `WM_NULL` message. So, EurekaLog will detect a hang.

Note: operating system does not send `WM_NULL` messages to your threads. OS doesn't need this, because it already have all information available (information about last sent message and delay times). However, EurekaLog has no access to this information - thus, it must send `WM_NULL` message to detect hangs.

This technique works only in GUI applications (the same as technique used by operating system) and only for main thread (because GUI in VCL, CLX and FMX applications are restricted to main thread).

However, if your particular application allow some way to detect hangs - you may use `RaiseFreezeException` function to trigger hang detection. For example, if you spawn a background thread (to offload heavy work and let GUI remain responsive), and if you did not get reply from your background thread in sane amount of time - then you can consider your background thread as hanged, and you can call `RaiseFreezeException` function to invoke freeze detection dialog.

If your application is running on Vista+ system (e.g. Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, etc.) - then EurekaLog will use Wait Chain Traversal (WCT) API to detect deadlocks between threads. Live locks are not detected.

Once EurekaLog detects hang or deadlock in application - it raises a special constructed exception. This immediately triggers a standard exception processing, which invokes EurekaLog, displays a error dialog, sends report, etc.

Hangs - Developer Perspective

The operating system and EurekaLog defines an application hang as a UI thread that has not processed messages for at least 5 seconds (for OS) or 60 seconds (default for EurekaLog). Obvious bugs cause some hangs, for example, a thread waiting for an event that is never signaled, and two threads each holding a lock and trying to acquire the others. You can fix those bugs without too much effort. However, many hangs are not so clear. Yes, the UI thread is not retrieving messages - but it is equally busy doing other 'important' work and will eventually come back to processing messages.

However, the user perceives this as a bug. The design should match the user's expectations. If the application's design leads to an unresponsive application, the design will have to change. Finally, and this is important, unresponsiveness cannot be fixed like a code bug; it requires upfront work during the design phase. Trying to retrofit an application's existing code base to make the UI more responsive is often too expensive. The following design guidelines might help:

- Make UI responsiveness a top-level requirement; the user should always feel in control of your application
- Ensure that users can cancel operations that take longer than one second to complete and/or that operations can complete in the background; provide appropriate progress UI if necessary
- Queue long-running or blocking operations as background tasks (this requires a well-thought out messaging mechanism to inform the UI thread when work has been completed)
- Keep the code for UI threads simple; remove as many blocking API calls as possible
- Show windows and dialogs only when they are ready and fully operational. If the dialog needs to display information that is too resource-intensive to calculate, show some generic information first and update it on the fly when more data becomes available. A good example is the folder properties dialog from Windows Explorer. It needs to display the folder's total size, information that is not readily available from the file system. The dialog shows up right away and the "size" field is updated from a worker thread

Unfortunately, there is no standard simple way to design and write a responsive application. Windows and Delphi do not provide a simple asynchronous framework that would allow for easy scheduling of blocking or long-running operations. The following sections introduce some of the best practices in preventing hangs and highlight some of the common pitfalls. However, there are some 3rd party frameworks and solutions available, which can help you with developing smooth applications. Please look for information about AsyncCalls, TasksEx and OTL.

Best Practices

Keep the UI Thread Simple

The UI thread's primary responsibility is to retrieve and dispatch messages. Any other kind of work introduces the risk of hanging the windows owned by this thread.

Do:

- Move resource-intensive or unbounded algorithms that result in long-running operations to worker threads
- Identify as many blocking function calls as possible and try to move them to worker threads; any function calling into another DLL should be suspicious
- Make an extra effort to remove all file I/O and networking API calls from your worker thread. These functions can block for many seconds if not minutes. If you need to do any kind of I/O in the UI thread, consider using asynchronous I/O
- Be aware that your UI thread is also servicing all single-threaded apartment (STA) COM servers hosted by your process; if you make a blocking call, these COM servers will be unresponsive until you service the message queue again

Do not:

- Wait on any kernel object (like Event or Mutex) for more than a very short amount of time; if you have to wait at all, consider using `MsgWaitForMultipleObjects`, which will unblock when a new message arrives
- Share a thread's window message queue with another thread by using the `AttachThreadInput` function. It is not only extremely difficult to properly synchronize access to the queue, it also can prevent the Windows operating system from properly detecting a hung window
- Use `TerminateThread` on any of your worker threads. Terminating a thread in this way will not allow it to release locks or signal events and can easily result in orphaned synchronization objects
- Call into any 'unknown' code from your UI thread. This is especially true if your application has an extensibility model; there is no guarantee that 3rd-party code follows your responsiveness guidelines
- Make any kind of blocking broadcast call; `SendMessage(HWND_BROADCAST)` puts you at the mercy of every ill-written application currently running

Implement Asynchronous Patterns

Removing long-running or blocking operations from the UI thread requires implementing an asynchronous framework that allows offloading those operations to worker threads.

Do:

- Use asynchronous window message APIs in your UI thread, especially by replacing SendMessage with one of its non-blocking peers: PostMessage, SendNotifyMessage, or SendMessageCallback
- Use background threads to execute long-running or blocking tasks. Use the new thread pool API to implement your worker threads
- Provide cancellation support for long-running background tasks. For blocking I/O operations, use I/O cancellation, but only as a last resort; it's not easy to cancel the 'right' operation

Use Locks Wisely
Your application or DLL needs locks to synchronize access to its internal data structures. Using multiple locks increases parallelism and makes your application more responsive. However, using multiple locks also increases the chance of acquiring those locks in different orders and causing your threads to deadlock. If two threads each hold a lock and then try to acquire the other thread's lock, their operations will form a circular wait that blocks any forward progress for these threads. You can avoid this deadlock only by ensuring that all threads in the application always acquire all locks in the same order. However, it isn't always easy to acquire locks in the 'right' order. Software components can be composed, but lock acquisitions cannot. If your code calls some other component, that component's locks now become part of your implicit lock order - even if you have no visibility into those locks.

Things get even harder because locking operations include far more than the usual functions for Critical Sections, Mutexes, and other traditional locks. Any blocking call that crosses thread boundaries has synchronization properties that can result in a deadlock. The calling thread performs an operation with 'acquire' semantics and cannot unblock until the target thread 'releases' that call. Quite a few User32 functions (for example SendMessage), as well as many blocking COM calls fall into this category.

Worse yet, the operating system has its own internal process-specific lock that sometimes is held while your code executes. This lock is acquired when DLLs are loaded into the process, and is therefore called the 'loader lock.' The DllMain function always executes under the loader lock; if you acquire any locks in DllMain (and you should not), you need to make the loader lock part of your lock order. Calling certain Win32 APIs might also acquire the loader lock on your behalf - functions like LoadLibraryEx, GetModuleHandle, and especially CoCreateInstance.

Do:

- Design a lock hierarchy and obey it. Add all the necessary locks. There are many more synchronization primitives than just Mutex and CriticalSections; they all need to be included. Include the loader lock in your hierarchy if you take any locks in DllMain
- Agree on locking protocol with your dependencies. Any code your application calls or that might call your application needs to share the same lock hierarchy
- Lock data structures not functions. Move lock acquisitions away from function entry points and guard only data access with locks. If less code operates under a lock, there is less of a chance for deadlocks
- Analyze lock acquisitions and releases in your error handling code. Often the lock hierarchy if forgotten when trying to recover from an error condition
- Replace nested locks with reference counters - they cannot deadlock. Individually locked elements in lists and tables are good candidates
- Be careful when waiting on a thread handle from a DLL. Always assume that your code could be called under the loader lock. It's better to reference-count your resources and let the worker thread do its own cleanup (and then use FreeLibraryAndExitThread to terminate cleanly)
- Use the Wait Chain Traversal if you want to diagnose your own deadlocks

Do not:

- Do anything other than very simple initialization work in your DllMain function. Especially do not call LoadLibraryEx or CoCreateInstance
- Write your own locking primitives. Custom synchronization code can easily introduce subtle bugs into your code base. Use the rich selection of operating system and Delphi's RTL synchronization objects instead
- Do any work in the constructors and destructors for global variables, they are executed

under the loader lock

Be Careful with Exceptions
Exceptions allow the separation of normal program flow and error handling. Because of this separation, it can be difficult to know the precise state of the program prior to the exception and the exception handler might miss crucial steps in restoring a valid state. This is especially true for lock acquisitions that need to be released in the handler to prevent future deadlocks.

Do:
- Use try/finally pattern with locks to ensure releasing lock on exceptions
- Be careful with the code executing in exception handler; the exception might have leaked many locks, so your handler should not acquire any

Do not:
- Handle native exceptions if not necessary or required. If you use native exception handlers for reporting or data recovery after catastrophic failures, consider using the EurekaLog or default operating system mechanism of Windows Error Reporting instead

This article is based on Preventing Hangs in Windows Applications

Part VIII

8 Customizing EurekaLog

Note: this is advanced article on customizing EurekaLog. Novice and beginners can start with Basic procedures 45 tutorial instead.

EurekaLog 7 offers many ways for customizing behavior and visual appearance (error dialogs).

You can use one or more of the following methods (all methods are listed starting with most high-level/easy-to-use to low-level/hard-to-use):

1. EurekaLog options 180:
   a. Static options 180
   b. Dynamic options 183:
      i. Variables 183
      ii. Filters 185
2. Coding 189:
   a. Changing options at run-time 189
   b. Custom attributes 190
   c. Events 192
   d. Subclassing 195
   e. Low-level handlers 211
   f. Modifying code of EurekaLog itself 212

See also:
- Customizing feedback 155

8.1 EurekaLog options

EurekaLog options is the most simple way to affect on EurekaLog 180. They were specifically designed for that purpose. Most customizations can be done by using options:
- Static options 180
- Dynamic options 183

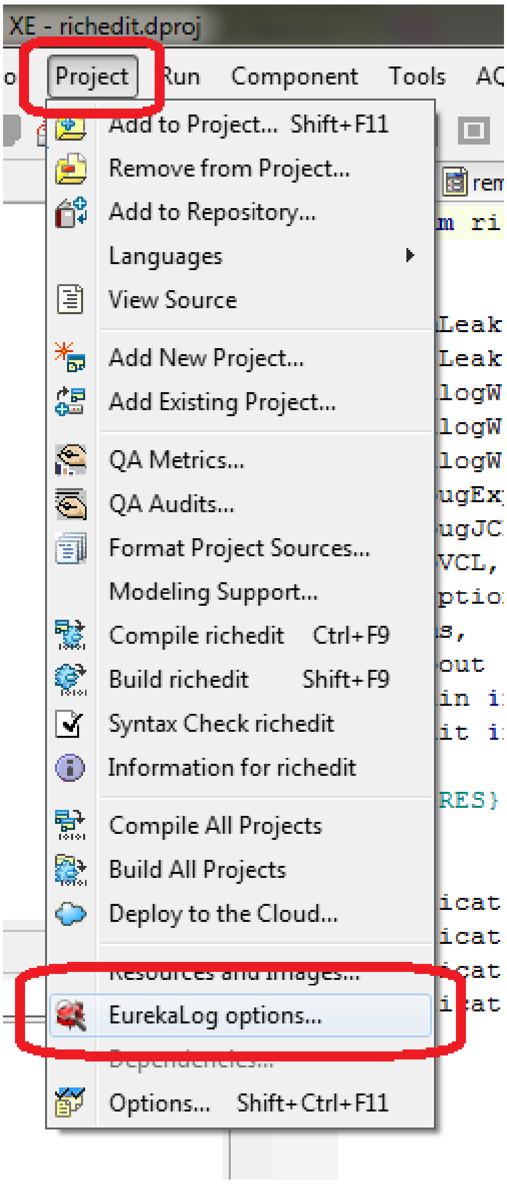
See also:
- Coding 189

8.1.1 Static options

EurekaLog stores per-project configuration in options file for your project (it's .dproj/.cbproj for new Delphi/C++ Builder versions). Each time you build your project with EurekaLog - these options are inserted into final .exe file (or DLL). Then EurekaLog's code can use these options at run-time.

Therefore, you can set options during design-time - and they will be accounted when application is run. In other words, EurekaLog project options works exactly as standard project options dialog.

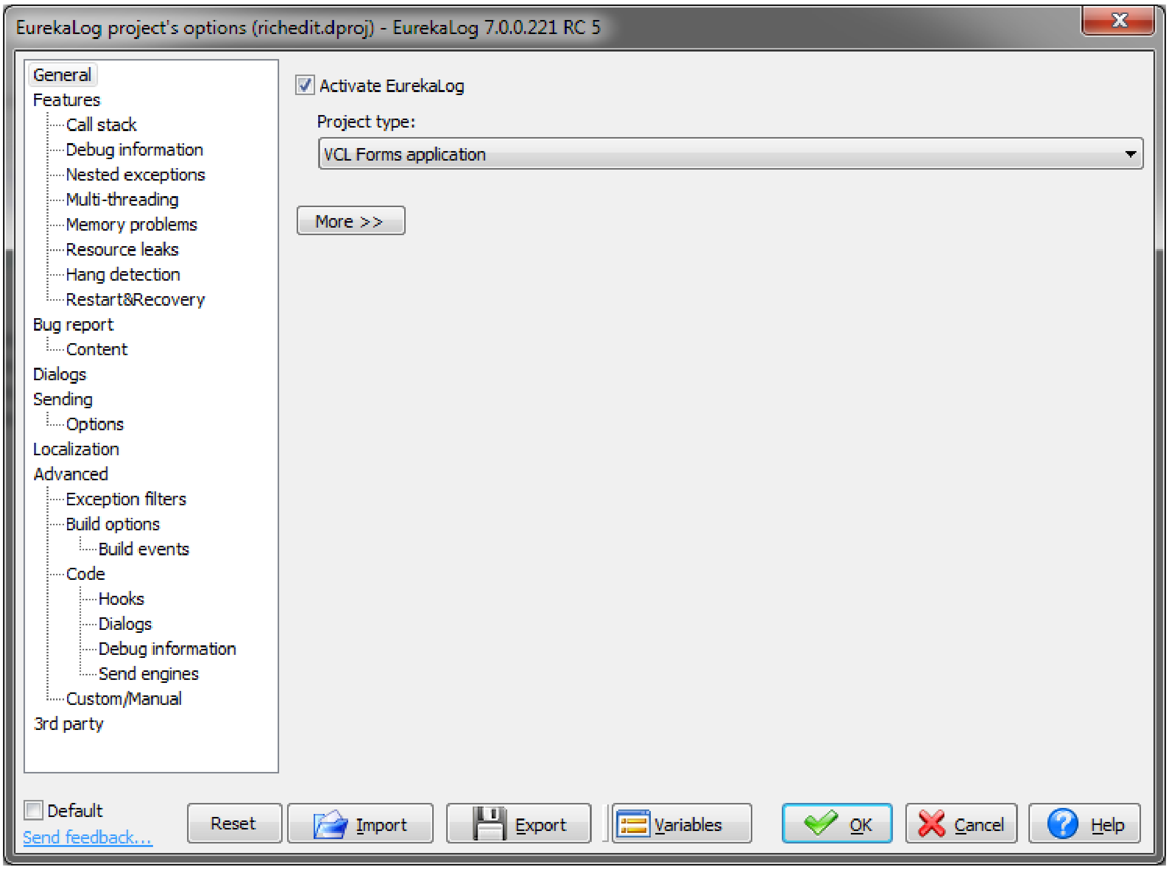
To view/edit options - you must first open a project in your IDE. Then you should use "Project"/"EurekaLog options" menu command:



Opening EurekaLog project options

(You can also invoke this command if you don't have opened project, but then you'll edit default options for each new project).

The dialog itself is similar to project options dialog from Delphi/C++ Builder:

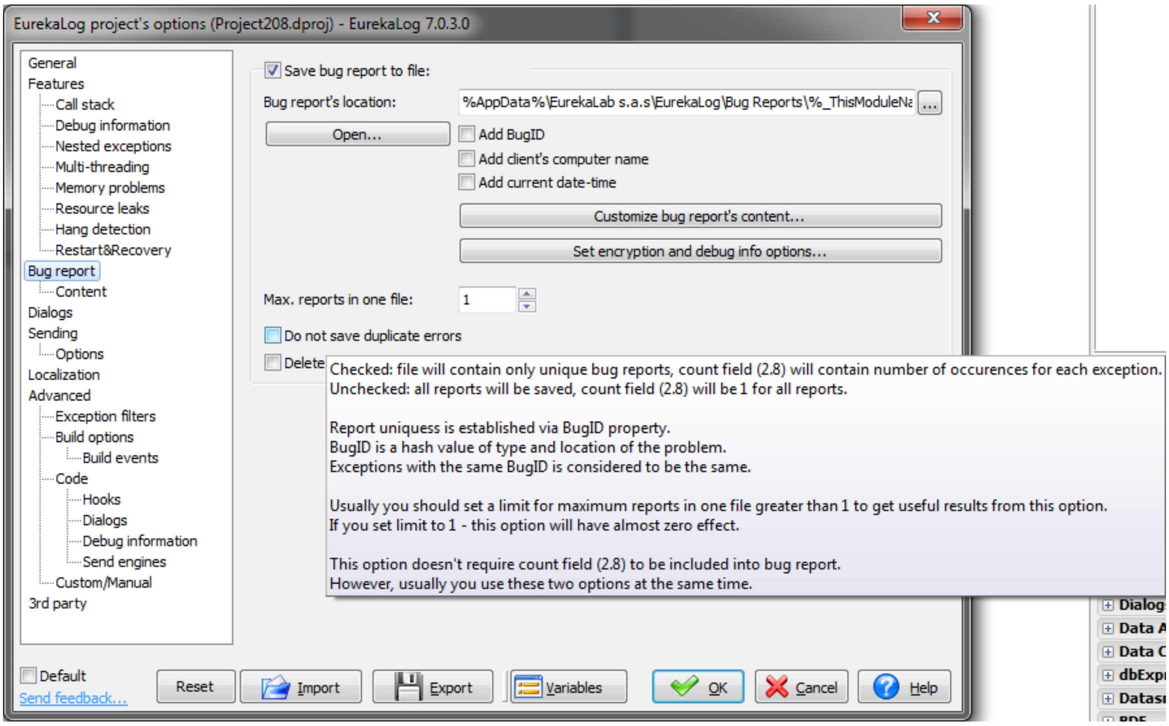


EurekaLog project options

You can see options categories on the left part (TreeView). Click on any category or sub-category to view and/or change options inside this category.

With this dialog you may change EurekaLog behaviour - to save bug report to disk or not, and where to save it, and what will be inside bug report, and should it be sent to you (as developer), and many other things.

Options are preset to fulfil common needs of developers. As you may guees, not always these "common needs" is what you specifically need in your own project. So we suggest you just to walk through options dialog to get yourself familiar with available options and alter them as you need. Each options has a detailed description in our help system. Yes, F1 works, if you've installed local help files during installation of EurekaLog (which is by default). But you may always visit our online help or just press F1 225. Moreover, each option has short explanation in the hint. Just hover option's control with mouse cursor and wait a little:



☑ Save bug report to file:

Bug report's location: %AppData%\EurekaLab s.a.s\EurekaLog\Bug Reports\%_ThisModuleNa

□ Add BugID

□ Add client's computer name

□ Add current date-time

Max. reports in one file: 1

□ Do not save duplicate errors

□ Delete

Checked: file will contain only unique bug reports, count field (2.8) will contain number of occurences for each exception.
Unchecked: all reports will be saved, count field (2.8) will be 1 for all reports.

Report uniquess is established via BugID property.
BugID is a hash value of type and location of the problem.
Exceptions with the same BugID is considered to be the same.

Usually you should set a limit for maximum reports in one file greater than 1 to get useful results from this option.
If you set limit to 1 - this option will have almost zero effect.

This option doesn't require count field (2.8) to be included into bug report.
However, usually you use these two options at the same time.

Popup hint with explanation of the option under the mouse cursor

Conclusion:
- + easy to use, no coding is necessary
- - options are static (i.e. set at design time)

See also:
- Dynamic options 183
- Project options 225
- External options 443
- Customizing EurekaLog 180

8.1.2 Dynamic options

There are two additional features to mitigate "static" negative effect 180 of options:
- Variables 183
- Filters 185

8.1.2.1 Variables

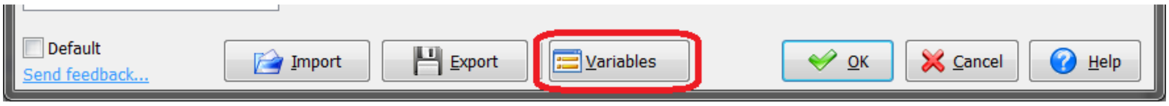
Environment variables 413 are a set of dynamic named values that can affect the way running processes will behave on a computer. They can be said in some sense to create the operating environment in which a process runs. For example, an environment variable with a standard name can store the location that a particular computer system uses to store temporary files - this may vary from one computer system to another. A process which invokes the environment variable by (standard) name can be sure that it is storing temporary information in a directory that exists and is expected to have sufficient space.

You can use environment variables in any text values in your project settings. You can insert variable by using "Variables" window. Variable is inserted as special tag. When you run your application at run-time, any variable value will be replaced with actual value, which is calculated at run-time.

For example, if you set your folder for saving bug-report to "`%APPDATA%\MyBugReports`" then your bug reports will be saved to `C:\Users\`*`UserName`*`\AppData\Roaming\MyBugReports\` or `C:\Documents and Settings\`*`UserName`*`\Application Data\MyBugReports` - depending on real value of `%APPDATA%` variable at run-time.

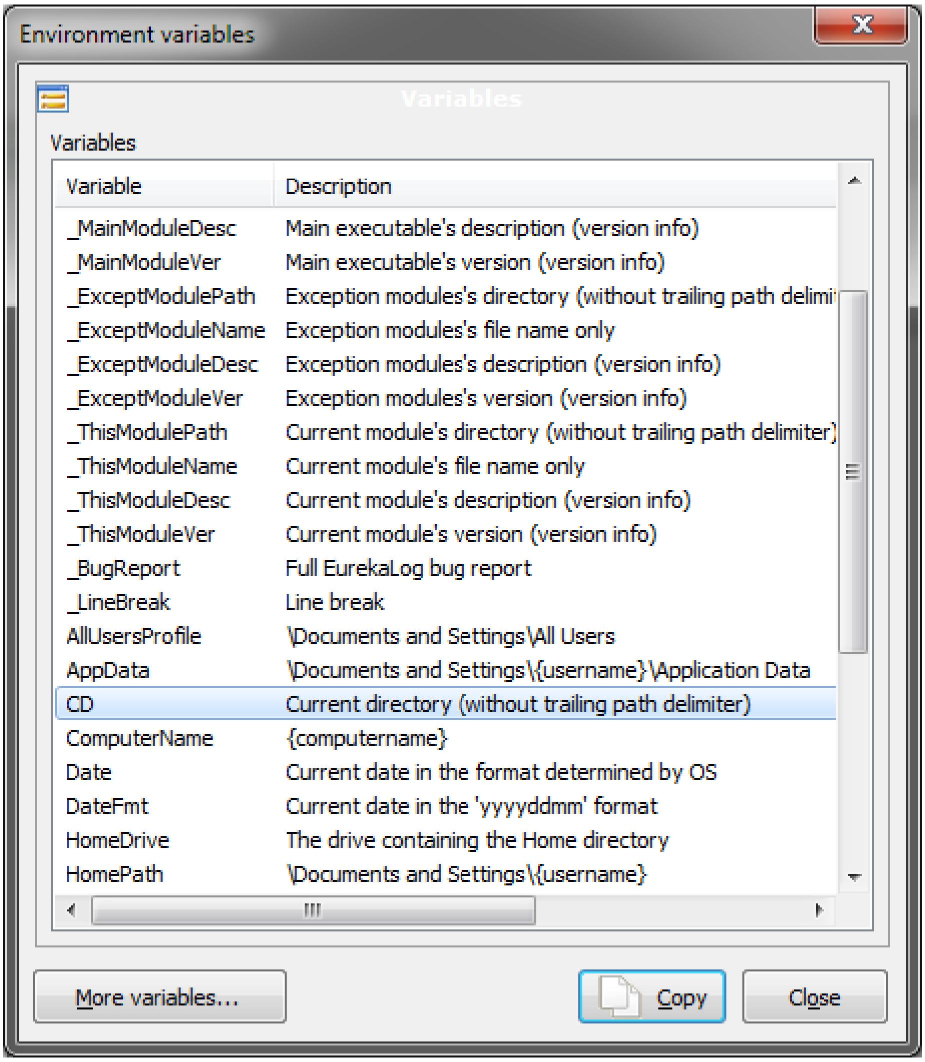
Note: variables names are case-insensitive.

When you're in your EurekaLog project options (see above), you can click on "Variables" button at dialog's bottom:



"Variables" button in EurekaLog project options

Click on this button and you will see such window:



Variable	Description
_MainModuleDesc	Main executable's description (version info)
_MainModuleVer	Main executable's version (version info)
_ExceptModulePath	Exception modules's directory (without trailing path delimi
_ExceptModuleName	Exception modules's file name only
_ExceptModuleDesc	Exception modules's description (version info)
_ExceptModuleVer	Exception modules's version (version info)
_ThisModulePath	Current module's directory (without trailing path delimiter)
_ThisModuleName	Current module's file name only
_ThisModuleDesc	Current module's description (version info)
_ThisModuleVer	Current module's version (version info)
_BugReport	Full EurekaLog bug report
_LineBreak	Line break
AllUsersProfile	\Documents and Settings\All Users
AppData	\Documents and Settings\{username}\Application Data
CD	Current directory (without trailing path delimiter)
ComputerName	{computername}
Date	Current date in the format determined by OS
DateFmt	Current date in the 'yyyyddmm' format
HomeDrive	The drive containing the Home directory
HomePath	\Documents and Settings\{username}

"Variables" dialog

"Copy" button will close the window with copying selected variable into clipboard. Alternatively: you can just double-click on variable in the list.
"Close" button will close the window without any action.

Suggested action's sequence:
1. Open "Variables" window.
2. Select variable that you want to use.
3. Click "Copy" (or double-click variable). Dialog will be closed.
4. Paste variable name from clipboard (Ctrl + V or Shift + Ins) to target setting's edit box.

Notes:

- This dialog suggests you only build-in special pseudo-variables (those with names started with "_") and commonly used variables. However, you can use any environment variable (even if it's not listed in this dialog).
- You can also use relative file paths. For example, any path which starts with . (dot) will be relative to your current executable (regardless of actual current folder, which may be changed, say, by system open dialog). For example, ".\BugReport.el" means file in the same folder as your executable.

See more:

- Explanation of environment variables (with the list of available variables) 413

So, while variables is an easy way to add some dynamic behaviour to EurekaLog options, their powers are limited. There may be no variable for your specific need or it may be not applicable to desired options (such as checked/unchecked option kind). In this case - you can set up/alter options from code 189.

See also:

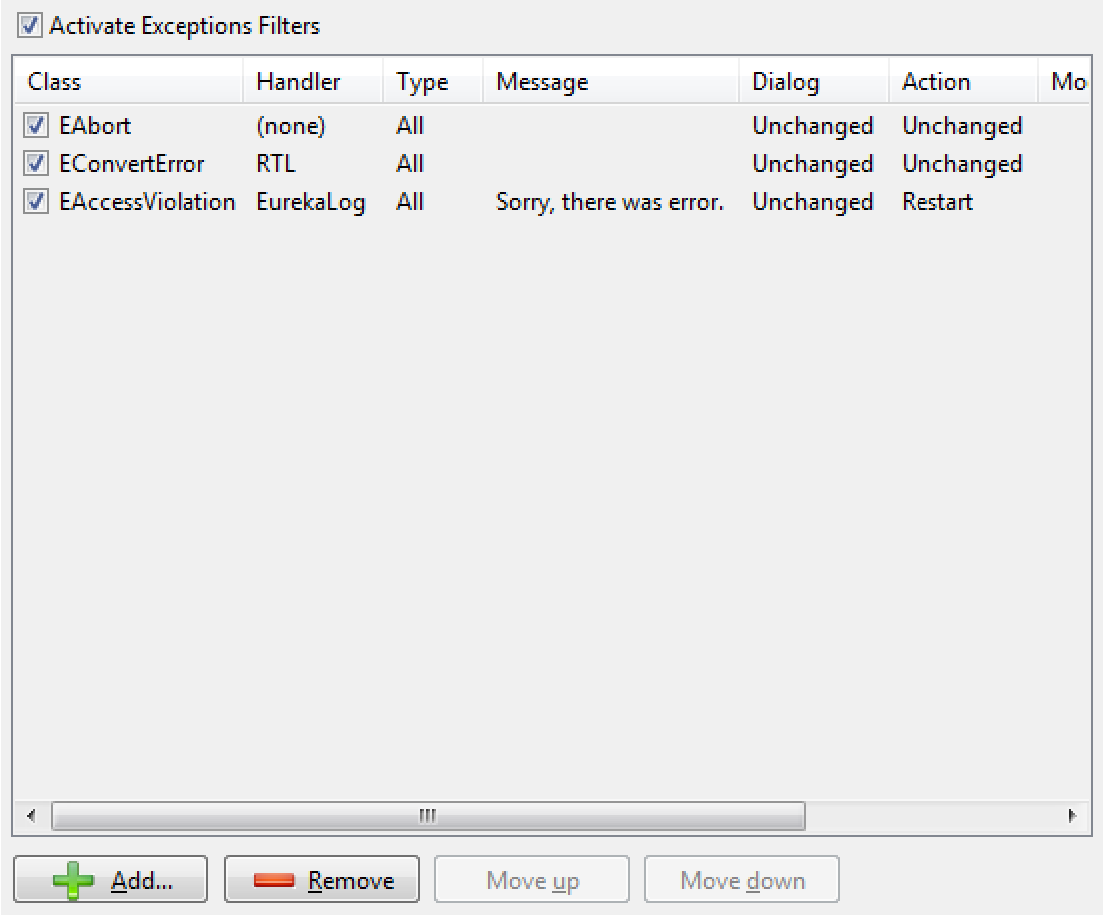
- Filters 185
- Customizing EurekaLog 180
- Coding 189

8.1.2.2 Filters

Exception filter is a filter which can alter EurekaLog's behavior based on some properties of exception. It is a easy way to customize EurekaLog on per-exception basis without writing code. Exception filters can be configured here 343.

Usually exceptions are identified by exception's class name (such as 'Exception', 'EAccessViolation', 'EStreamReadError', etc.). You can also identify exception by source location or values of properties. And you can identify exception by its type (handled or unhandled). Once exception is identified - you can change handler for it (none, RTL or EurekaLog), dialog class (to any of existing dialog classes), override error message, set action (restart or terminate) or mark exception as "expected". Remember, that you can also use environment variables.

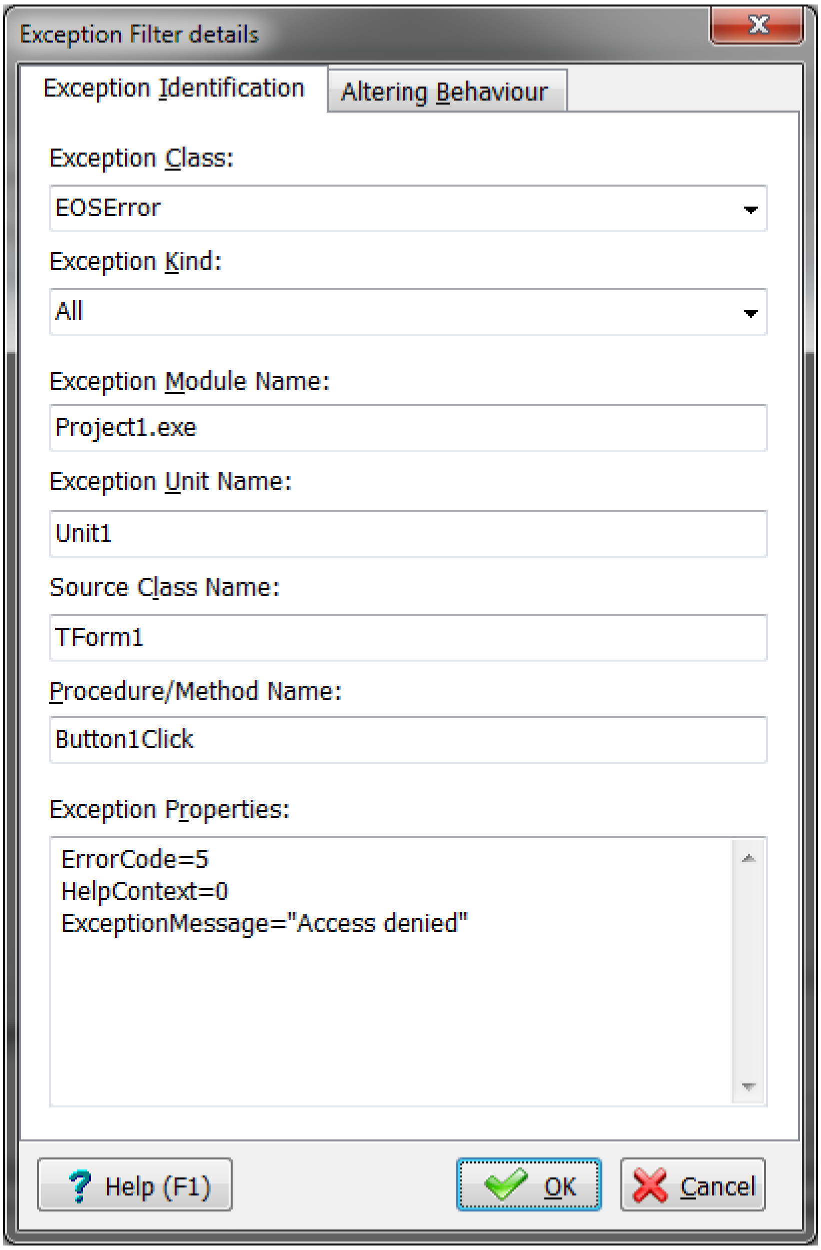
Exception filters are applied before processing exception. Filters are applied in order from top to bottom. First matched filter is applied.



☑ Activate Exceptions Filters

Class	Handler	Type	Message	Dialog	Action	Mo
☑ EAbort	(none)	All		Unchanged	Unchanged	
☑ EConvertError	RTL	All		Unchanged	Unchanged	
☑ EAccessViolation	EurekaLog	All	Sorry, there was error.	Unchanged	Restart	

Example of 3 exceptions filters added to options



Exception Class:
EOSError

Exception Kind:
All

Exception Module Name:
Project1.exe

Exception Unit Name:
Unit1

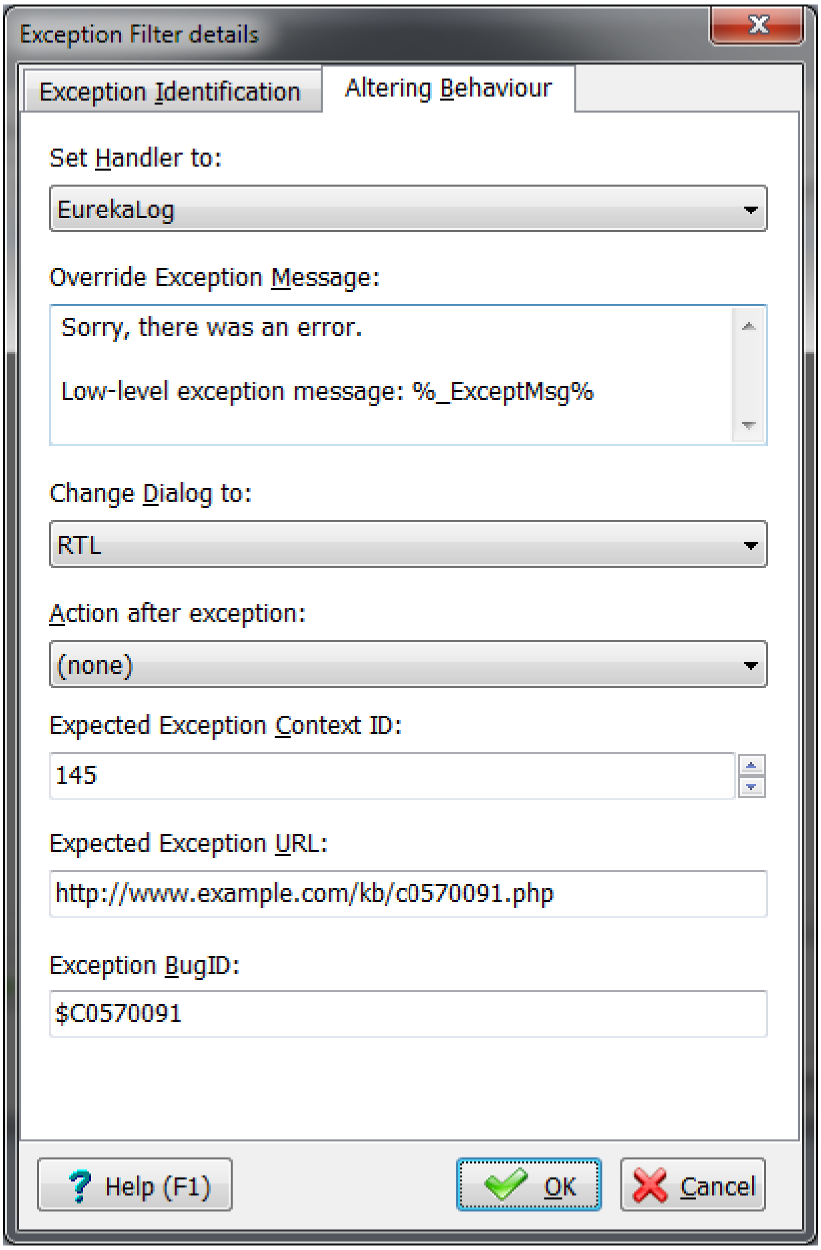
Source Class Name:
TForm1

Procedure/Method Name:
Button1Click

Exception Properties:

```
ErrorCode=5
HelpContext=0
ExceptionMessage="Access denied"
```

Exception Identification Page



Exception Filter details

Set Handler to: EurekaLog

Override Exception Message:

Sorry, there was an error.

Low-level exception message: %_ExceptMsg%

Change Dialog to: RTL

Action after exception: (none)

Expected Exception Context ID: 145

Expected Exception URL: http://www.example.com/kb/c0570091.php

Exception BugID: $C0570091

Altering Behavior Page

Most typical usage for exceptions filters include:

- Completely ignore particular exception type. This is useful for such exceptions as EAbort. You may use similar exception types in your code, so now you can use exception filters feature to completely hide such exceptions. Set handler to "none". No error dialog will be shown. No bug report will be created.
- Excluding certain exceptions from EurekaLog's processing. For example, when disk free space is low and you try to create or write a file - an exception about insufficient disk space will be raised. You usually want this exception to be shown as error to the end user (so he/she can free some disk space and retry the operation), rather than generating bug report (and optionally sending it to you). That's because this exception is not a bug in your code, there is nothing to fix. So you may create filter for such exceptions (they are called "expected" exceptions). Set handler to RTL, and this exception will be handled as usual (by showing error message), but no EurekaLog work will be done for it (no EurekaLog error dialog, no bug reports, etc.).
- Some exceptions (such as access violation) are low-level. Their message contains some technical information, but it's completely useless for most users (which are not programmers). Seeing such "cryptic" error messages may be confusing ("gosh, what should I do with it? Is it even a error? Or it asks me something that I don't understand?"),

so it may be useful to hide them. It's better to show more user-friendly message like "Sorry, there was a error in application. Please, let us know about this problem and restart application." Such message is far more descriptive. It explains what happenned (an error) and what to do (report to developers and restart application). You may create exception filters for such exceptions, set handler to "EurekaLog" and override a error message. You may additionally enable termination/restart features for these exceptions. **Note:** message override affects only error message in visual dialogs. Bug reports always contain information about original exception (with original error message).

There may be other use for exception filters (just use your imagination). But the above are the most popular ones.

Important note: your exception classes must be real classes. You can not use aliases. For example:

```
type
  EMyException1 = class(Exception);
  EMyException2 = Exception;
```

You can specify 'EMyException1' as exception class name. However, 'EMyException2' will not work. Because there is no such class. You should use 'Exception' instead.

Important note: ensure that features specified in exception filters will be available at run-time. For example, if your application uses MS Classic-styled exception dialog by default and you want to switch to EurekaLog-styled dialog for some exceptions by using exception filters - then be sure to include code for EurekaLog dialog into your application 354.

Note: exceptions filter can also be created from code 189. Also, exception filters can be replaced with custom attributes 190.

See also:

- Exception filters options 343
- Variables 183
- Customizing EurekaLog 180
- Coding 189

8.2 Coding

While the previously discussed methods 180 do not require writing code and provide some degree of dynamic changes at run-time 183, they are still limited and not suitable for custom, not common cases. So, when you have specific need that can't be satisfied by the discussed methods - then you'll have to write some code:

- Changing options at run-time 189
- Custom attributes 190
- Events 192
- Subclassing 195
- Low-level handlers 211
- Modifying code of EurekaLog itself 212

See also:

- Customizing EurekaLog 180

8.2.1 Changing options at run-time

One of the simplest customizations is changing EurekaLog options 180 from the code. ExceptionLog7 unit declares global function CurrentEurekaLogOptions. This function returns TEurekaModuleOptions class (which is declared in EClasses unit), which contains properties corresponding to EurekaLog's options. I.e. almost each property (that you can change in EurekaLog project options dialog 225) is presented as property in TEurekaModuleOptions class.

For example, you may write a program which is able to work either as Win32 service or as

GUI front end - depending on the way it was launched. You probably want to have different bug reports for each of this mode. And Win32 service can't have visual error dialogs. Obviously, this can't be solved by setting options at design-time, since trigger event happens at run-time. So, you setup common options at design time and write such code, which sets the differences:

```
uses
  ETypes, EClasses, ExceptionLog7;

...

initialization

  if IsWin32Service then
  begin
    CurrentEurekaLogOptions.OutputPath := '%APPDATA%\MySoftware\Win32Service\';
    CurrentEurekaLogOptions.ExceptionDialogType := edtService;
  end
  else
  begin
    CurrentEurekaLogOptions.OutputPath := '%APPDATA%\MySoftware\GUIFrontEnd\';
    CurrentEurekaLogOptions.ExceptionDialogType := edtMSClassic;
  end;

end.
```

Notes:

- Some options (such as build options 349) affects compilation stage. Thus, they can't be changed at run-time when file was already compiled.
- Some options (such as leaks control 250) requires special rules for setting up 508. That's why they are controlled with other means, not with options class. However, 90% of options can be freely changed via this class.
- Needless to say that you may use environment variables 183 and exception filters 185 from code too. You can alter exception filters via `CurrentEurekaLogOptions.ExceptionsFilters` property.

`CurrentEurekaLogOptions` is a global function which contains default options for your application in run-time. It affects all exceptions and all threads. Typically, you alter these options at startup (either in initialization section of some unit or in begin/end of .dpr file).

Often there is need to alter option only for some exceptions. EurekaLog provides additional feature for this: events and event hanlders 192.

Important note: ensure that features specified by your code will be available at run-time. For example, if your application uses MS Classic-styled exception dialog by default and you want to switch to EurekaLog-styled dialog with your code - then be sure to include code for EurekaLog dialog into your application 354. The same is true for send engines 355, Hooks 352 and debug information providers 355 are registered on startup and could not be customized at run-time.

See also:

- Coding 189
- Customizing EurekaLog 180
- Configuring project for leaks detection 508

8.2.2 Custom attributes

New Delphi and C++ Builder IDEs offer extented RTTI with support of custom attributes. EurekaLog is able to use custom attributes to alter behaviour for exception types. This is similar to exception filters 185. Using exception filters is simple, but sometimes it's not very convenient. So, instead you can define EurekaLog behaviour right when you declare exception classes (this code will be ignored if there is no EurekaLog installed).

Custom attributes are declared in EClasses unit. The following attributes are declared:

```
type
  EurekaLogHandler = class(TCustomAttribute)
  strict private
    FHandler: TFilterHandlerType;
  public
    constructor Create(Value: TFilterHandlerType);
    property Handler: TFilterHandlerType read FHandler;
  end;

  EurekaLogMessage = class(TCustomAttribute)
  strict private
    FMessage: String;
  public
    constructor Create(Value: String);
    property Message: String read FMessage;
  end;

  EurekaLogAction = class(TCustomAttribute)
  strict private
    FAction: TFilterActionType;
  public
    constructor Create(Value: TFilterActionType);
    property Action: TFilterActionType read FAction;
  end;

  EurekaLogDialog = class(TCustomAttribute)
  strict private
    FDialog: TExceptionDialogType;
  public
    constructor Create(Value: TExceptionDialogType);
    property Dialog: TExceptionDialogType read FDialog;
  end;

  EurekaLogExpected = class(TCustomAttribute)
  strict private
    FURL: String;
    FContext: Integer;
    FBugID: TBugID;
  public
    constructor Create(const AContextNumber: Integer = -1;
      const AURL: String = ''; const ABugID: TBugID = 0);
    property URL: String read FURL;
    property Context: Integer read FContext;
    property BugID: TBugID read FBugID;
  end;
```

Here is a sample on how to use them:

```
type
  // ETestException will be ignored by EurekaLog and
  // it always will be handled by your application
  // (as if EurekaLog would be disabled)
  [EurekaLogHandler(fhtRTL)]
  ETestException = class(Exception);

  // Exception message will be replaced
  [EurekaLogMessage('Sorry, there was an error.')]
  ECustomMessage = class(Exception);

  // Switch dialog to EurekaLog-Detailed style
  [EurekaLogDialog(edtEurekaLogDetailed)]
  EDialogException = class(Exception);

  // Mark exception as expected
  [EurekaLogExpected()]
  EMyException1 = class(Exception);

  // Mark exception as expected, assign help topic ID = 1234,
  // show "Help" button in dialogs
  [EurekaLogExpected(1234)]
  EMyException2 = class(Exception);

  // Mark exception as expected, assign online help topic (URL),
  // show "Help" button in dialogs
  [EurekaLogExpected(0, 'http://www.example.com/kb/low_memory.asp')]
  EMyException3 = class(Exception);

  // Mark exception as expected, assign fixed BugID
  [EurekaLogExpected(-1, '', $C4F10001)]
  EMyException4 = class(Exception);
```

You get the idea - when you declare exception type, you can add attributes to it. Each attribute will modify EurekaLog behavior for exceptions of this class only. This works the same as exception filters 185 (actually, internally custom attributes just creates new filter rule).

To learn more about custom attributes - visit RAD Studio help.

Important note: ensure that features specified by your code will be available at run-time. For example, if your application uses MS Classic-styled exception dialog by default and you want to switch to EurekaLog-styled dialog with your code - then be sure to include code for EurekaLog dialog into your application 354. The same is true for send engines 355. Hooks 352 and debug information providers 355 are registered on startup and could not be customized at run-time.

See also:

- Filters 185
- Coding 189
- Customizing EurekaLog 180

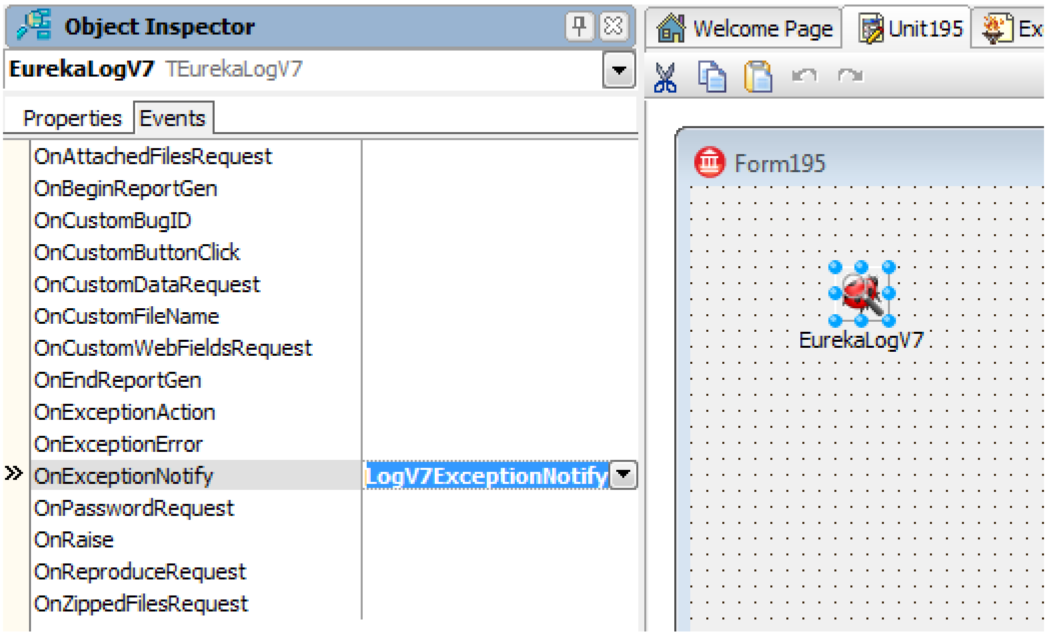
8.2.3 Events

EurekaLog's event is similar to standard Delphi/C++ Builder events, such as OnClick for TButton. Event handler is callback procedure or method, which you regiter for some event. EurekaLog will call your event handler (i.e. your code) each time this event occurs. There are many events available. Most commonly used are `OnExceptionNotify` and `OnExceptionAction`.

Important note: event handlers will not be called during leaks processing. That is because leaks detection happens AFTER finalizing application. See Configuring project for leaks detection 508 for more info.

Note: each event handler is declared as both method or procedure. You can use either option, there is no difference between method and procedure, so you can use whatever you want.

There are two ways of registering your code as event handler. The most simple way is to use TEurekaLogEvents component. This component is similar to standard TApplicationEvents component. Just drop it on the form and assign handlers by using "Events" page in Object Inspector:


Object Inspector

EurekaLogV7 TEurekaLogV7

Properties | Events

OnAttachedFilesRequest
OnBeginReportGen
OnCustomBugID
OnCustomButtonClick
OnCustomDataRequest
OnCustomFileName
OnCustomWebFieldsRequest
OnEndReportGen
OnExceptionAction
OnExceptionError
OnExceptionNotify — LogV7ExceptionNotify
OnPasswordRequest
OnRaise
OnReproduceRequest
OnZippedFilesRequest

TEurekaLogEvents component on the form and one event hanlder assigned

While this is very simple way, you should be aware that your event handlers will be registered as long as host form lives. Thus, your event handler will not be invoked for events during initialization/finalization of your application. For this reason:

- While there is no limitation on `TEurekaLogEvents` components in one application and no restrictions on the form (it can be main form and any child/auxilary form), we recommend to use main form as host for `TEurekaLogEvents` component whenever possible.
- We recommend using component only for such events that depends on your host form. For example, event hanlder may add currently opened document to bug report. Such event handler is best registered with component. We don't recommend using component for more generic event handlers. Use code registration (see below).

Note: registering event handler via component will create/use event handler as method of the object (i.e. form), not as callback procedure.

Second method to register event handler is to call `RegisterEvent`*`Name`* function, where *Name* is a name of event. For example, you may use such code:

```
unit Unit1;

interface

...

implementation

uses
  EEvents;

// Some your routine to add a message to application log
procedure AddToLog(const AMessage: String);
begin
  ...
end;

// Your handler for OnExceptionNotify event
procedure MyHandler(const ACustom: Pointer; AExceptionInfo: TEurekaExceptionInfo;
  var AHandle: Boolean; var ACallNextHandler: Boolean);
begin
  if AExceptionInfo.ExceptionClass = 'EMyException' then
  begin
    AddToLog(AExceptionInfo.CallStack.ToString);
    AHandle := False;
  end;
end;

initialization

  RegisterEventExceptionNotify(nil, MyHandler);

end.
```

This sample event handler will log call stack of each exception of EMyException type, and it will disable EurekaLog's processing for these exceptions.

As you can see - you need to write all code by yourself (i.e. handler registration and procedure header/prototype). All events are declared in EEvents unit.

Note: you may unregister registered event handler by using UnregisterEvent*Name* function, but it's not required.

That's almost all for events and event handlers, but we must discuss one more important thing. Once exception is raised, EurekaLog options are captured from CurrentEurekaLogOptions function. Each exception is accompanied by TEurekaExceptionInfo class, which contains property and options for each exception. Further changes in CurrentEurekaLogOptions will not affect already raised exceptions. Threfore, your changes of CurrentEurekaLogOptions inside any event handler for exceptions will have no effect for current exception. It will affect only future exceptions. To change options of current exceptions - use Options property of TEurekaExceptionInfo class. For example:

```
unit Unit1;

interface

...

implementation

uses
  EEvents;

// Your handler for OnExceptionNotify event
procedure MyHandler(const ACustom: Pointer; AExceptionInfo: TEurekaExceptionInfo;
  var AHandle: Boolean; var ACallNextHandler: Boolean);
begin
  // The same as exception filter for EMyException which changes dialog to "EurekaL
  if AExceptionInfo.ExceptionClass = 'EMyException' then
    AExceptionInfo.Options.ExceptionDialogType := edtEurekaLog;
end;

initialization

  RegisterEventExceptionNotify(nil, MyHandler);

end.
```

The similar concern is applied for many other objects in EurekaLog. Such as dialogs, send engines, log report builders, call stack classes, etc. They all have `Options` property, which is captured from their "parent" or `CurrentEurekaLogOptions` if there is no "parent".

To summarize:

- Event hanlders offer you unlimited ability for customizing EurekaLog, but only within certain points of interest (i.e. events). You can't alter whole processing logic 195 by using only event handlers.

Important note: ensure that features specified by your code will be available at run-time. For example, if your application uses MS Classic-styled exception dialog by default and you want to switch to EurekaLog-styled dialog with your code - then be sure to include code for EurekaLog dialog into your application 354. The same is true for send engines 355, Hooks 352 and debug information providers 355 are registered on startup and could not be customized at run-time.

See also:

- List of available events
- Coding 189
- Customizing EurekaLog 180
- Configuring project for leaks detection 508

8.2.4 Subclassing

When event handlers 192 are not enough, you need to override parts of EurekaLog's code to fulfill your needs. For example, if you want to alter icon in error dialogs or replace dialog as whole, if you want to change processing (exchange order of saving bug report and sending it), etc. or if you want to extend EurekaLog (like add new dialog or new send method) - then you can't use event handlers, because there is no event for that. But what you can do is to replace or extend EurekaLog's code with your own.

High level code of EurekaLog is done as **classes**. Therefore, if you need to alter minor point in EurekaLog (adjust it a little), then you may create your own class as child class to standard EurekaLog class, override some virtual method, write your own behavior code, call inherited method to get standard behavior, and so on. I.e. the usual things that you may do with tree of classes. Or you may write your own class completely from scratch (inheriting from base abstract class).

Note: please note that EurekaLog classes are constantly evolving (unlike events). There are new methods and features introduced as time passes. Therefore your code that uses sub-classing may need to be adjusted for new EurekaLog versions.

For that reason classes are described much less in documentation than other methods. Because they are used much rarely and changes more often. To study what you can do with classes you will have to read interface sections of units with declaration of classes. There you can see class inheritance, interesting methods to override, etc. You can do this even in EurekaLog's editions without full source code. We provide .pas headers for your reference (see "Source" folder of your EurekaLog installation).

The common rule is: if class can be only one (like log builder class), then class type is registered via global class variable. All you have to do is to write your own class and assign it to variable during initialization. Otherwise, if there may be various classes (like dialogs or send engines), then you have to create your own class, register it in global list via various `Register`*`Something`* functions (they are declared in the same place as base abstract classes), and switch EurekaLog to use your class via options.

Let's see some code examples to make things clear.

Example #1: replacing dialog icon.

EurekaLog offers you feature to use generic error icon or icon of your application for error dialogs. However, you may want to use some other icon for that. There is no such feature in EurekaLog, but you can easily fix that with such code:

```
unit Unit1;

interface

// ...

implementation

uses
  EBase, ECore, EModules, EListView, EDialog, EDialogWinAPIMSClassic;

{$R *.dfm}

// To test our customization code
procedure TForm1.Button1Click(Sender: TObject);
begin
  raise Exception.Create('Error Message');
end;

type
  // Our child class - inheriting from standard TMSClassicDialog
  TMSClassicDialogCustom = class(TMSClassicDialog)
  private
    FCustomIcon: HBITMAP; // our new icon
  protected
    // Init/done:
    procedure WindowInit; override;
    procedure WindowDone; override;
    // Replacing icon drawing:
    function Paint(const ADC: HDC; const ARect: TRect): Integer; override;
  end;

{ TMSClassicDialogCustom }

procedure TMSClassicDialogCustom.WindowInit;
var
  Ico: HIcon;
begin
  inherited;

  Ico := LoadIcon(HInstance, 'CUSTOMICON');
  FCustomIcon := IcoToBmp(Ico, GetStockObject(WHITE_BRUSH), 32, 32);
end;

function TMSClassicDialogCustom.Paint(const ADC: HDC; const ARect: TRect): Integer;
begin
  Result := inherited;
  DrawBmp(ADC, FCustomIcon, MonitorLeft, MonitorTop, 32, 32);
end;

procedure TMSClassicDialogCustom.WindowDone;
begin
  DeleteObject(FCustomIcon);
  inherited;
end;

initialization
```

```
  if IsEurekaLogInstalled then
  begin
    // You have to register dialog before using it:
    RegisterDialogClass(TMSClassicDialogCustom);
    // Once registered - now you can use it in options:
    CurrentEurekaLogOptions.ExceptionDialogType := TMSClassicDialogCustom.ClassName
  end;

end.
```

Obviously, this code replaces icon for one particular dialog type (MS Classic style). If you want to alter icon for several classes, then you have to write more your own child classes.

Example #2: replacing content of bug reports.

Recently we got a request from a customer. They use their own system to collect and sort bug reports generated by EurekaLog. Previously they worked with EurekaLog 6, but then upgraded to EurekaLog 7. Bug reports from EurekaLog 7 has slightly different structure than reports from EurekaLog 6 (new fields and columns). Therefore, new format of bug report file breaks their old code. Of course, they can update/improve their code or... switch EurekaLog to produce bug reports in old format. This can also be useful if you're migrating to EurekaLog from other exception tracers solutions (such as JCLDebug/JCLHookExcept, madExcept or Exception Magic). You can force EurekaLog to generate bug report in the format of your previous solution, so you won't have to re-write other code. Obviously, there is no such build-in feature in EurekaLog.

Here is a simple code that creates bug report in format of EurekaLog 6:

```
uses
  EConsts, ETypes, EClasses, ECallStack, EException, EEvents, ELogBuilder;

type
  // EurekaLog 6 format will be altered from default EurekaLog 7 format,
  // that's why we inherit from EurekaLog 7 builder class and
  // replace only some methods
  TEurekaLog6Builder = class(TLogBuilder)
  public
    function CreateGeneralText: String; override;
    function CreateCallStackText: String; override;
  end;

{ TEurekaLog6Builder }

// Replace header of the bug report (to change "7" to "6" in header)
function TEurekaLog6Builder.CreateGeneralText: String;
begin
  Result := inherited;
  Result := 'EurekaLog 6' + Copy(Result, Pos(sLineBreak, Result), MaxInt);
end;

// Replace call stack (remove new columns)
function TEurekaLog6Builder.CreateCallStackText: String;
var
  Stack: TEurekaCallStack;
  Formatter: TEurekaBaseStackFormatter;
begin
  Stack := nil;
  try
    if CallStack <> nil then
    begin
      Stack := TEurekaCallStack.Create;

      Stack.Assign(CallStack);

      Stack.Formatter.Assign(Options);
      Stack.Formatter.CaptionHeader :=
        Options.CustomizedExpandedTexts[mtDialog_CallStackHeader] + EHeaderSuffix;
    end;

    Formatter := TEurekaStackFormatterV6.Create;
    try
      Formatter.Assign(Options);
      Formatter.CaptionHeader :=
        Options.CustomizedExpandedTexts[mtDialog_CallStackHeader] + EHeaderSuffix;

      Result := CallStackToString(Stack,
        Options.CustomizedExpandedTexts[mtDialog_CallStackHeader] + EHeaderSuffix,
        Formatter);
    finally
      FreeAndNil(Formatter);
    end;
  finally
    FreeAndNil(Stack);
  end;
end;

// Rename .el files to old .elf files
procedure CustomizeFileNames(const ACustom: Pointer; AExceptionInfo: TEurekaExcepti
  const AFileType: TEurekaLogFileType; var AFileName: String; var ACallNextHandler:
begin
  if AnsiLowerCase(ExtractFileExt(AFileName)) = '.el' then
    AFileName := ChangeFileExt(AFileName, '.elf');
end;

initialization

  // Register bug report builder:
  LogBuilderClass := TEurekaLog6Builder;

  // Register event handler for file names:
  RegisterEventCustomFileName(nil, CustomizeFileNames);

end.
```

This sample uses a combination of custom class and event handler to reach the desired goal.

Example #3: override original class.

Suppose that you want to alter original behavior/class, but don't want to create a new class (with new name), because this will require you to alter options as well. Instead, you want just to set up options at design time and provide altered behavior at run-time.

For example, when you post bug to Mantis, a issue title is composed. EurekaLog does not provide way to alter it, because it's used to identify tickets. You may want to alter it (say, by appending more info, so title becomes more descriptive). Here is how you can do that:

```
uses
  EConsts,
  ESend,
  ESendAPIMantis;

type
  // Trick: use the same class name as original class
  // You'll have to append unit name to class ident to avoid compiler confusion
  TELTrackerMantisSender = class(ESendAPIMantis.TELTrackerMantisSender)
  protected
    function ComposeTitle: String; override;
  end;

// This is a default implementation of the method,
// you can replace it with arbitrary code
function TELTrackerMantisSender.ComposeTitle: String;
begin
  if BugAppVersion <> '' then
    Result := Format('%s (Bug %s; v%s)', [BugType, BugID, BugAppVersion])
  else
    Result := Format('%s (Bug %s)', [BugType, BugID]);
  Log(Format('Title = ''%s''', [Result]));
end;

initialization

  // Register send class to be the first in the list.
  // Default class (by EurekaLog) will be listed later.
  // Any search for class by name will find our class, because it's listed first
  RegisterSenderFirst(TELTrackerMantisSender); // <- Notice "First" in the name

end.
```

With this trick you don't have to change options. You can just set options at design-time. The important part here is to register your class first, which is archived by using registering function with "First" suffix.

Note: please note that this is not recommended way to work with tickets in Mantis. We suggest you to create custom fields for your project (this is done in Mantis configuration). Fill custom fields with information (this is done in EurekaLog configuration). And show some of these fields in list of tickets (this is done in Mantis configuration). That way you will archive the desired effect (to see more info about each ticket in list), but also additionally gain some benefits: you will be able to sort/filter by custom fields, you will not break default EurekaLog tickets identification.

Example #4: a custom dialog.

You may be not satisfied by standard EurekaLog dialogs, so you may want to use your own dialog. Here is what you need to do: create class inheriting from abstract `TBaseDialog` class (`EDialog` unit), register it, and set `ExceptionDialogType` option. This example is similar to example #1. But this time we create dialog from scratch, we're not using ready classes.

The following sample shows four new dialog classes. Actually, this is just a code from EurekaLog itself, but it's short and simple to illustrate the point:

```
uses
  EDialog, EClasses, ETypes;

type
  // "Empty" dialog that does nothing at all
  TNullDialog = class(TBaseDialog)
  protected
    procedure Beep; override;
    function ShowModalInternal: TResponse; override;
  public
    class function ThreadSafe: Boolean; override;
  end;

  // MessageBox dialog
  TMessageBoxDialog = class(TBaseDialog)
  protected
    function ShowModalInternal: TResponse; override;
    procedure Beep; override;
  public
    class function ThreadSafe: Boolean; override;
  end;

  // A variant of MessageBox with more detailed message (with call stack)
  TMessageBoxDetailedDialog = class(TMessageBoxDialog)
  protected
    function ExceptionMessage: String; override;
  end;

  // "Default" dialog - dialog that invokes standard dialog (non-EurekaLog)
  TRTLHandlerDialog = class(TBaseDialog)
  protected
    procedure Beep; override;
    function GetCallRTLExceptionEvent: Boolean; override;
    function ShowModalInternal: TResponse; override;
  end;

{ TNullDialog }

procedure TNullDialog.Beep;
begin
  // does nothing - no beep needed
end;

// Main method: do nothing, return success
function TNullDialog.ShowModalInternal: TResponse;
begin
  SetReproduceText(ReproduceText);

  Result.SendResult := srSent;
  Result.ErrorCode := ERROR_SUCCESS;
  Result.ErrorMessage := '';
end;

// Indicate that we can be called from any thread
// (this should be False for VCL/CLX/FMX dialogs)
class function TNullDialog.ThreadSafe: Boolean;
begin
  Result := True;
end;

{ TMessageBoxDialog }

procedure TMessageBoxDialog.Beep;
begin
  // does nothing - beep is invoked by Windows.MessageBox in
  // TMessageBoxDialog.ShowModalInternal
end;

// Main method
function TMessageBoxDialog.ShowModalInternal: TResponse;
var
  Flags: Cardinal;
  Msg: String;
begin
  // Set default result
  Result.ErrorCode := ERROR_SUCCESS;
  Result.ErrorMessage := '';
  if SendErrorReportChecked then
    Result.SendResult := srSent
  else
    Result.SendResult := srCancelled;

  // Prepare message to show
  Msg := ExceptionMessage;
  if ShowSendErrorControl then
  begin
    Msg := Format(Options.CustomizedExpandedTexts[mtSend_AskSend], [Msg]);
    Flags := MB_YESNO;
  end
  else
    Flags := MB_OK;
  Flags := Flags or MB_ICONERROR or MB_TASKMODAL;
  if SendErrorReportChecked or (not ShowSendErrorControl) then
    Flags := Flags or MB_DEFBUTTON1
  else
    Flags := Flags or MB_DEFBUTTON2;

  // Call actual MessageBox and set result
  case MessageBox(Msg,
                  Options.CustomizedExpandedTexts[mtDialog_Caption],
                  Flags) of
    0: Result.ErrorCode := GetLastError;
    IDYes:
       Result.SendResult := srSent;
    IDNo:
       Result.SendResult := srCancelled;
  end;

  // Save error code/error message for failures
  if Result.ErrorCode <> ERROR_SUCCESS then
  begin
    Result.SendResult := srUnknownError;
    Result.ErrorMessage := SysErrorMessage(Result.ErrorCode);
  end
  else
    SetReproduceText(ReproduceText);
end;

// Can be called from any thread
class function TMessageBoxDialog.ThreadSafe: Boolean;
begin
  Result := True;
```

```
end;

{ TRTLHandlerDialog }

// Indicate desire to invoke RTL handler
function TRTLHandlerDialog.GetCallRTLExceptionEvent: Boolean;
begin
  Result := True;
end;

function TRTLHandlerDialog.ShowModalInternal: TResponse;
begin
  SetReproduceText(ReproduceText);

  Result.SendResult := srRestart;  // means "call RTL handler"
  Result.ErrorCode := ERROR_SUCCESS;
  Result.ErrorMessage := '';
end;

procedure TRTLHandlerDialog.Beep;
begin
  // Does nothing - transfer work to RTL handler
end;

{ TMessageBoxDetailedDialog }

// This one is a bit more complex - we want to add call stack to error message.
// However, default form is not very readable with variable-width fonts.
// That's why first we need a way to format call stack in another way.

type
  // Our new formatter
  TMessageBoxDetailedFormatter = class(TEurekaBaseStackFormatter)
  protected
    function GetItemText(const AIndex: Integer): String; override;
    function GetStrings: TStrings; override;
  end;

// Forms one line of call stack
function TMessageBoxDetailedFormatter.GetItemText(const AIndex: Integer): String;
var
  Cache: TEurekaDebugInfo;
  Info: PEurekaDebugInfo;
  ModuleName, UnitName, RoutineName, LineInfo: String;
begin
  Info := CallStack.GetItem(AIndex, Cache);

  ModuleName := ExtractFileName(Info^.Location.ModuleName);
  UnitName := Info^.Location.UnitName;

  if UnitName = ChangeFileExt(ModuleName, '') then
    UnitName := ''
  else
    UnitName := '.' + UnitName;

  RoutineName := CallStack.ComposeName
```

```
    (Info^.Location.ClassName, Info^.Location.ProcedureName);
  if RoutineName <> '' then
    RoutineName := '.' + RoutineName;

  if Info^.Location.LineNumber > 0 then
    LineInfo := Format(',%d[%d]',
      [Info^.Location.LineNumber, Info^.Location.ProcOffsetLine])
  else
    LineInfo := '';

  Result := ModuleName + UnitName + RoutineName + LineInfo;
end;

// Formats entire call stack
function TMessageBoxDetailedFormatter.GetStrings: TStrings;
var
  ThreadID: Cardinal;
  I: Integer;
  Line: String;
  Stack: TEurekaBaseStackList;
begin
  if not Assigned(FStr) then
  begin
    FStr := TStringList.Create;
    FModified := True;
  end;
  if FModified then
  begin
    Stack := CallStack;
    CalculateLengths;
    FStr.BeginUpdate;
    try
      FStr.Clear;
      FStr.Capacity := Stack.Count;

      if Stack.Count > 0 then
      begin
        ThreadID := Stack.Items[0].ThreadID;
        for I := 0 to Stack.Count - 1 do
        begin
          if (Stack.Items[I].Location.Module <> 0) and
             (Stack.Items[I].Location.DebugDetail in [ddUnit..ddSourceCode]) and
             (Stack.Items[I].ThreadID = ThreadID) then
          begin
            Line := GetItemText(I);
            if (FStr.Count <= 0) or (FStr[FStr.Count - 1] <> Line) then
              FStr.Add(Line);
          end;
        end;
      end;
    finally
      FStr.EndUpdate;
    end;
    FModified := False;
  end;
  Result := FStr;
end;

// Append call stack to error message
function TMessageBoxDetailedDialog.ExceptionMessage: String;
const
  MaxLines = 15;
var
```

```
  Formatter: TMessageBoxDetailedFormatter;
  Stack: TEurekaBaseStackList;
begin
  {$WARNINGS OFF}
  // Abstract methods are intended here.
  // It is like assert: they should not be called.
  Formatter := TMessageBoxDetailedFormatter.Create;
  {$WARNINGS ON}
  try

    if Assigned(CallStack) then
      Formatter.Assign(CallStack.Formatter);
    Formatter.CaptionHeader := '';

    Stack := nil;
    try
      if CallStack <> nil then
      begin
        Stack := TEurekaStackList.Create;
        Stack.Assign(CallStack);
        while Stack.Count > MaxLines do
          Stack.Delete(Stack.Count - 1);
      end;
      Result := inherited ExceptionMessage + sLineBreak + sLineBreak +
                CallStackToString(Stack, '', Formatter);
    finally
      FreeAndNil(Stack);
    end;
  finally
    FreeAndNil(Formatter);
  end;
end;

...

initialization

  RegisterDialogClass(TNullDialog);
  RegisterDialogClass(TMessageBoxDialog);
  RegisterDialogClass(TMessageBoxDetailedDialog);
  RegisterDialogClass(TRTLHandlerDialog);

end.
```

As you can see, the central method here is ShowModalInternal. It does all the work. It's abstract and **must** be overwritten in child classes. All other methods of TBaseDialog are virtual, but not abstract. They contain default behavior. You can override them to alter behavior, but you don't have to. Base dialog class contains large number of helpers (methods and properties). All that dialog needs to do is to invoke these methods in right order. Therefore any child class can use powerful tools to quickly build new dialog.

Note: there is another abstract dialog class - TWinAPIDialog from EDialogWinAPI unit. It's useful if you want to create new dialog based on direct WinAPI calls, rather than using ready functions or frameworks (VCL/CLX/FMX).

Important note: dialog class is responsible for almost whole exception processing. That's because "dialog" don't have to be visual. Think about Win32 service, system log, WER (Windows Error Reporting), etc. So, this is not always possible to distinguish between "error dialog" and "exception processing". That's why these concepts are both controlled by single "dialog" class. As we saw above, a major method for *visual* dialog is ShowModalInternal method. But real entry point is Execute method. Default implementation goes like this:

```
function TBaseDialog.Execute: TResponse;
var
  CanSaveReport: Boolean;
begin
  try
    SaveCurrentEnvironment;
    try
      SetupFileNames;
      FDuplicate := CalcDuplicatedException(FCanSend);

      if not Restarted then
      begin
        MakeScreenshot;
        AddCustomData;
        if DeleteLogAtVersionChange then
          DeleteOldLog;
        SetReproduceText('');
        if PresaveReport and SaveLogFile then
          SaveBugReport; // Pre-save to get log in case of crash in dialog
        Beep;
      end
      else
        FSaved := (PresaveReport and SaveLogFile);
      Result := ShowModal;

      // Restart dialog?
      if Result.SendResult = srRestart then
        Exit;

      // Save bug report
      CanSaveReport := SaveLogFile and
                       (
                         Succeeded(Ord(Result.SendResult)) or
                         (Result.SendResult = srCancelled) or
                         (not PresaveReport)
                       );
      if CanSaveReport then
        SaveBugReport; // Re-save to update changed fields (like reproduce text)

      if Succeeded(Ord(Result.SendResult)) and CanSend then
      begin
        PrepareFilesForSend;

        Result := SendBugReport;

        if Failed(Ord(Result.SendResult)) then
        begin
          if CopyLogInCaseOfError then
            CopyReportToClipboard;
          if SaveCompressedCopyInCaseOfError then
            SavePackedCopy;
        end
        else
        if DeleteLogAfterSuccessfulSend then
          DeleteCurrentLog;

        ShowSendResult(Result);
        if CanSaveReport then
          UpdateSendInformationInLog(Result.SendResult);
      end
      else
      begin
        DoEventExceptionAction(ExceptionInfo, atSendCancelled);
        if CanSaveReport then
          UpdateSendInformationInLog(srCancelled);
      end;

    finally
      RestoreCurrentEnvironment;
    end;

    CheckTermination;
    if Options.CustomFieldBool[difTerminateApplication] then
      TerminateApplication;
  except
    on E: Exception do
    begin
      Result.ErrorCode := ERROR_GEN_FAILURE;
      Result.ErrorMessage := E.Message;
      Result.SendResult := srUnknownError;
    end;
  end;
end;
```

As you can see, there is the whole processing of exceptions: saving bug report, displaying dialog, updating bug report with new values (e-mail/steps to reproduce), sending report, restarting application. You rarely need to alter this method, rather you will override methods which are called by it. This method is shown here just to illustrate the point that dialog controls more than just visual behavior.

As final words on sub-classing - here is the list of classes, units and functions for this (as of EurekaLog 7.0.02):

- Base dialog class: `EDialog.TBaseDialog`.
- Dialog class registration: `EDialog.RegisterDialogClass`.
- Base send engine class: `ESend.TELUniversalSender`.
- Send engine class registration: `ESend.RegisterSender`.
- Base debug information provider class: `EClasses.TELDebugInfoSource`.
- Debug information provider registration: `EDebugInfo.RegisterDebugInfoSource`.
- Base call stack class: `ECallStack.TEurekaBaseStackList`.
- Default call stack class: `ECallStack.EurekaCallStackClass`.
- Specific call stack classes: `ECallStack.TracerMethodsClasses`.
- Base log builder class: `ELogBuilder.TBaseLogBuilder`.
- Default log builder class: `ELogBuilder.LogBuilderClass`.
- Base hung detection thread class: `EFreeze.TFreezeThread`.
- Default hung detection thread class: `EFreeze.FreezeThreadClass`.

See also:

- Events 192
- Coding 189
- Customizing EurekaLog 180

8.2.5 Low-level handlers

Classes 195 are used by high level code in EurekaLog. Sometimes customizations of high level code is not enough. Luckily, EurekaLog 7 offers you a way to change almost anything. Low level points for customizations are presented by global variables of procedural types. These procedural variables are invoked by EurekaLog, and it points to EurekaLog code by default. You may write your own code and write pointer to it into these variables.

Note: these global variables are not documented and never will be. You can use them on your own risk.

Here is the list of available low-level customization possibilities:

- EInject unit:
  - EventUnhandledExceptionFilter
  - EventExceptProc
  - EventExceptClsProc
  - EventExceptObjProc
  - EventRaiseExceptionProc
  - EventGetExceptionStackInfoProc
  - EventGetStackInfoStringProc
  - EventCleanUpStackInfoProc
  - EventExceptionDispatcher
  - EventIndyThreadHandleException
- EExceptionHook unit:
  - _IsUnexpected
- ELowLevel unit:
  - _InitDoneErrorHandler
  - _SafeExecLog
  - _ConsoleLogger
- EHook unit:
  - _GetSymbolAddr
- EBase unit:
  - _InternalError
  - _AllowBypassInternalErrors
  - _RaiseExpected
  - _OnPanic
- EAppType unit:
  - _CustomMessageBox
  - _GetMainWnd
- Almost all routines in EMemLeaks and EResLeaks units.
- EThreadsManager unit:
  - _GetRealAddresses

There are some other points to override default handlers, but they are almost useless for customizations, since they are used for very specific EurekaLog's needs.

The most important thing about low level handlers is that the initialization order is important. It's highly not recommended to alter these variables during actual work of your application. You should modify handlers only during initialization/finalization process. It's also strictly recommended to save previous handler before installing your own handler and revert it back (uninstall) on finalization.

See also:

- Coding 189
- Customizing EurekaLog 180
- Configuring project for leaks detection 508
- Internal errors 591

8.2.6 Modifying code of EurekaLog itself

When nothing else helps 211 - you can alter source code of EurekaLog. Of course, it's possible only if you have edition with full source code (all other methods of customizations listed previously 180 are applicable for any edition of EurekaLog).

You can alter EurekaLog code to absolutely every behavior that you want. Then you need to recompile it for these changes to take effect. See also: how to recompile EurekaLog 619.

Please note that any upgrading or reinstalling EurekaLog will overwrite EurekaLog files, thus your customizations will be lost (unless you made a backup). You'll have to re-implement all customizations each time you upgrade EurekaLog. For this reason use this method as last resort measure only. Prefer the above discussed methods whenever possible. If you still has to modify EurekaLog sources - please, let us know why you do that. We can add your customizations in EurekaLog, so you won't need to re-insert them into source code each time you upgrade (of course, this is only possible if your customizations has some value for other developers).

Part IX

9 Frequently Asked Questions (FAQ)

This section covers some problems that are frequently encountered by users of EurekaLog.

The questions are organized in the following categories:

- General FAQ 215
- Default file's locations 217
- File formats 218

9.1 General FAQ

Question: What I need to make EurekaLog-enabled application?
Answer: Install EurekaLog 20, open your project, enable EurekaLog in options 33, build your application, done! Now you can get extensive information about problems in your application.

You may also want to adjust default application settings. However, our defaults were carefully choosen to be satisfying for most typical cases.

Question: Do I need to distribute any additional files for application compiled with EurekaLog?
Answer: No. EurekaLog-enabled applications are self-contained 38. All code and data are injected into executable module, so you don't need any additional files. No DLLs, no .map files, no .tds files.

Question: Does EurekaLog-enabled application require Internet connection?
Answer: No. You don't need Internet access to run your applications. You can disable sending bug reports and store all reports only locally (or even don't store at all - just show error dialogs).

Question: Do I need Internet connection for my developer machine?
Answer: No. You don't need Internet access on your machine either. EurekaLog do not check license online. EurekaLog installer is self-contained. You can also disable online checking for new EurekaLog version.

Question: How does EurekaLog work?
Answer: EurekaLog integrates itself with Delphi's and C++ Builder's IDE.
When you build your projects: EurekaLog injects all required data into your final executable modules.
When you run your application: EurekaLog installs hooks to catch raising of exceptions and memory allocations.
When exception is raised: EurekaLog collects information about exception.
When memory is allocated: EurekaLog collects information about memory block.
EurekaLog collects information about environment and builds bug reports on your request, when exception is handled or when application exits. EurekaLog uses injected data to build human-readable call stacks to show you address, module name, unit name, class name, routine name and line number. All this behaviour is FULLY customizable.

More details 38.

Question: But will EurekaLog work, if I compile project from command line? We have build script on our build server.
Answer: Yes. It's possibly to use EurekaLog with command-line compilation. Please refer to our documentation 421 for more information.

Question: How much does EurekaLog increase compiled file size?
Answer: That depends on your desired settings. There is a trade-off: better bug reports detalization require more information injected, smaller file size means less available

information. For example, you may want to include information for code from RTL and VCL; or you may want to exclude it and only supply information for your own code. Typical range of file size increase vary in range 1%-12%. Injected information is always packed and encrypted (EurekaLog uses ZLib and TEA).

See our guide on project settings 225.

Question: Can EurekaLog decrease an application's performance?
Answer: That depends on your application, but usually - no. That's because EurekaLog is not active in your application at all - until exception will be raised. When exception is raised - EurekaLog collects information. So, if your application raises a lot of exceptions - then yes, EurekaLog can slightly slow down its execution. Note that over-using of exceptions is a bad practice in general. You can solve this by disabling EurekaLog for particular exceptions or for particular code blocks.

There are also memory problems tracing features 250 in EurekaLog. You can disable them for best performance (it will be unaffected). If you enable such features - EurekaLog will collect information about memory blocks in your application. This can slow down your application. Exact estimates depends on your application (how often it allocates/releases memory). Typical slow down is about 5%.

Question: Does EurekaLog work with DLLs and BPLs?
Answer: Yes, it does. EurekaLog also have predefined templates of settings for DLLs and BPLs 363.

Question: Is there application type which EurekaLog do not support?
Answer: Probably no. We have templates (typical settings) for all common application types 363. And even if there is no template for your application type - you can just set up options manually. If there is no ready-to-use hook for your application type - you need to invoke EurekaLog from your global exception handler 370.

Question: Does EurekaLog work with COM-based applications?
Answer: Yes, it does. There is "Catch SafeCall exceptions" option 341, which is designed to catch exceptions in any COM object.

Question: What does EurekaLog do when it intercepts an error?
Answer: That depends on type of your application. EurekaLog can display error dialog (for GUI-based applications), EurekaLog can write text into console (for console-based applications), EurekaLog can silently log error to system log (for non-visual applications like Win32 services or web-server applications), EurekaLog can produce error HTML page (for web-applications). This behaviour can be fully customized. You can also use your own dialogs.

EurekaLog can also store bug report into file and send it to you (developer) via e-mail, HTTP or FTP upload, and submit to bug tracker software.

Question: Is EurekaLog compatible with .exe compression/protection software?
Answer: Yes, EurekaLog is FULLY compatible with .exe compression/protection software, so you can use those tools without any restrictions. EurekaLog have wide range of customization options.
There are options to keep .map file untouched 234 (if it's required by protection software).
There are options to disable hooks installation 352 (hooks installation can be detected as application's crack).
There are options to use hi-level or low-level hooks 349 (trade detalization for better compatibility).

Question: I do not want to expose my internal code information (routine names, etc.)
Answer: No problem! You can protect your information with password 243, you can exclude certain code blocks from inclusion (by using {$D-}/{$D+} compiler directives), or you can

include only unit names and line numbers 243 (but not class and routine names).

Question: What is the file's extension for bug reports?
Answer: It is *.EL for plain-text reports, *.ELP for packed reports (can include screenshot and additional files), *.ELX for XML reports. You can select desired format in options.

Question: How many errors can EurekaLog store in the file log?
Answer: There are no special limits.

Question: Can EurekaLog work in unattended environments?
Answer: Yes, there is no problem. You can disable visual feedback (error dialogs, send progress, etc.), you can set up logging, you can set up automatic bug report sending, you can set up auto-restart options.

Question: Can I ignore exception from 3rd party framework (some badly written code)?
Answer: Yes, of course. You can set up exception filter(s) 343 to ignore particular exception or you can disable EurekaLog for certain code blocks.

Question: What about locatization to other languages?
Answer: No problem. We support any localization tool or even no using localization tool at all. There are wide translation options available. You can localize resourcestrings, you can use Translate-like function (GetText-style) or you can just set up translated strings in options 339.

Question: Can I ask for feature to add? How about more information in bug reports?
Answer: Yes, sure - we love to hear feedback! You can ask us to add specific feature and additional information. We will add most popular and frequently asked features into next releases of EurekaLog. Unfortunately, not all features can be added. However, EurekaLog allows easy customization. There are many event handlers. So often you can add your desired features by yourself even without having full source code!

Oh, and we have OnCustomDataRequest event, which you can use to insert arbitrary information into bug reports. And you can also add additional files, of course.

Question: My question is not listed here?
Answer: Take a look at our video tutorials or code samples.

9.2 Default files names and locations

Short descrtiption

1. All working files (temp files) are placed in unique subfolder in %TEMP% folder.
2. All saved bug reports go to `%APPDATA%\Neos Eureka S.r.l\EurekaLog\Bug Reports`
3. A fallback packed bug report is saved to My documents folder.

Details

1. Working files

All working files (bug report, screen shot, packed report or XML report) are placed in unique subfolder in %TEMP% folder. Subfolder is deleted when exception processing will be done.

For example, a working bug report is stored at this path:
Windows XP: `C:\Documents and Settings\`*`UserName`*`\Local Settings\Application Data\Temp\{F62795F2-D5A2-4A5D-A90C-BB55A687F050}\BugReport.el`
Windows 7: `C:\Users\`*`UserName`*`\AppData\Local\Temp\{F62795F2-D5A2-4A5D-A90C-BB55A687F050}\BugReport.el`

{F62795F2-D5A2-4A5D-A90C-BB55A687F050} is random generated unique subfolder. It will be deleted after exception processing finishes.

Other working files are stored in the same folder, but named:

- BugReport.elp - packed bug report
- BugReport.xml - XML bug report
- BugReport.png - screenshot
- BugReport.mdmp - minidump
- BugReport.html - HTML page (web applications)

(see also 218)

Any additional files that may be saved in this folder will be deleted after exception processing finishes.

Those files are not created until needed.

2. Saved bug report
When "Save bug report to file" option is enabled, bug report is saved to this file:
Windows XP: `C:\Documents and Settings\`*`UserName`*`\Application Data\Neos Eureka S.r.l\EurekaLog\Bug Reports\Project1.exe\Project1.el`
Windows 7: `C:\Users\`*`UserName`*`\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Bug Reports\Project1.exe\Project1.el`
Local system account (same bitness): `C:\Windows\System32\config\systemprofile\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Bug Reports\Project1.exe\Project1.el`
Local system account (32-bit service on 64-bit system): `C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Bug Reports\Project1.exe\Project1.el`

Notes:
1. File name (the one without path) can be different, if additional options were enabled (like "Add computer name");
2. This folder can be customized in project's options 225. It's "Bug report's folder" option 264. Default value is "`%AppData%\Neos Eureka S.r.l\EurekaLog\Bug Reports\%_ThisModuleName%\`". Empty string means the same value. If you want to save bug report in the same folder as .exe file - you can use this value: "`.\`". You can also use any relative file path like this, for example: "`.\Reports\`".
3. If your custom save path will be write-protected at run-time, EurekaLog will revert it back to default (`%AppData%\Neos Eureka S.r.l\EurekaLog\Bug Reports\%_ThisModuleName%\`).

3. Send-failure report
When "Save bug report's copy in case of send failure" option is enabled and sending of bug report didn't work, bug report is saved to this file:
Windows XP: `C:\Documents and Settings\`*`UserName`*`\My documents\Project1.elp`
Windows 7: `C:\Users\`*`UserName`*`\Documents\Project1.elp`

See also:
- File formats 218

9.3 File formats

Actual formats in EurekaLog 7	
Extension	**Description**
.eof	Project's options (see also 227).
.eot	Localized texts (messages collection 339).
.el	Bug report in text format (see also 46).
.elx	Bug report in XML format (renamed XML).
.elp	Bug report in packed format (renamed ZIP).
.fdb	EurekaLog Viewer's default FireBird database.

.mdmp	Process mini-dump.
.html	HTML page (web applications).
Obsolete formats from EurekaLog 6	
Extension	**Description**
.elf	Bug report in text format.
.dat	The same as .elf. It was used as database by EurekaLog Viewer.
.xml	Bug report in XML format.
.zip	Bug report in packed format.

Part X

10 Integral parts

Here is a list of key parts of EurekaLog:
- IDE expert 221
- Command-line compiler 423 (advanced use only)
- Application types 363
- VCL components
- Dialogs 370
- Debug info providers 409
- Send engines 390
- Events
- Variables 413
- Tools 617

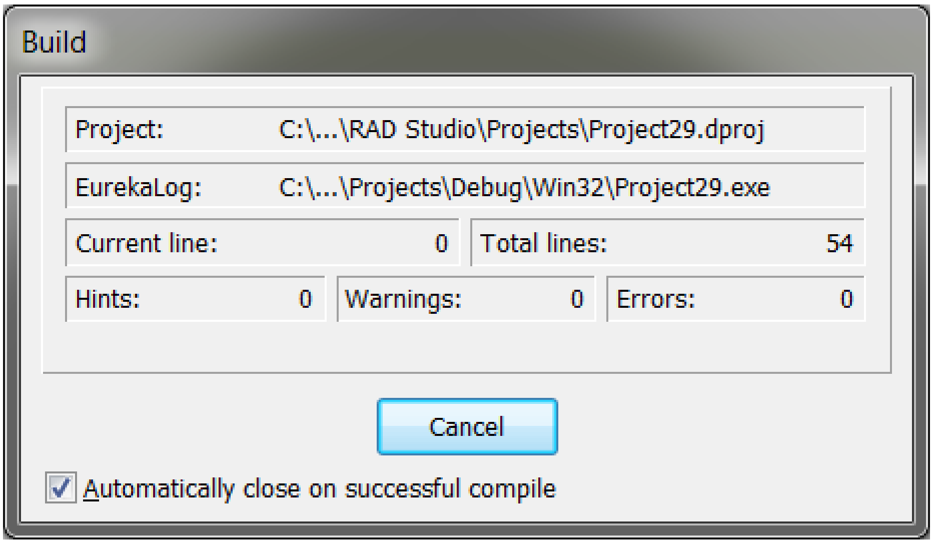
10.1 EurekaLog IDE expert

EurekaLog installs a .bpl packages into IDE. This adds a EurekaLog component and installs an **IDE expert**.

> "IDE expert" is a special term. IDE expert is a library or a package, which extends IDE by using special published interfaces. Those interfaces are called OpenTools API (OTA).

So, why EurekaLog needs that expert, and when you don't need it?

EurekaLog installs IDE expert to do the following:
- Adds menu items 222 to invoke EurekaLog.
- Allows you to edit global / IDE EurekaLog options 230.
- Allows you to edit project's EurekaLog options 225.
- Performs a post-processing of your project 421.



Project: C:\...\RAD Studio\Projects\Project29.dproj

EurekaLog: C:\...\Projects\Debug\Win32\Project29.exe

Current line: 0 | Total lines: 54

Hints: 0 | Warnings: 0 | Errors: 0

☑ Automatically close on successful compile

"EurekaLog" / "post-processing" stage during compilation. Notice "EurekaLog:" prefix on second line.

Don't see EurekaLog IDE expert working? 596

Now, back to the question: can you don't use IDE expert? **Yes** - you can uninstall IDE expert, if you don't want/need it. For example, you don't want to install EurekaLog on build's server. That's totally OK.

Here is what you can use instead:
- Adds menu items to invoke EurekaLog - you can add shortcuts to **Tools** menu manually, or you can run EurekaLog parts from the **Start** / **Programs** menu.
- Allows you to edit global / IDE EurekaLog options - you can use an external Debug

Symbols Setup tool 617 to setup global EurekaLog settings, which are not related to IDE.
- Allows you to edit project's EurekaLog options - you can use an external settings editor 617 to edit options in projects or .eof files.
- Performs a post-processing of your project - you can invoke a EurekaLog command-line compiler 423 to do post-processing manually 451. An alternative (easiest) way is to use MS-Build on Delphi 2007+ 429 or FinalBuilder 431.

See also:
- Compiling your project with EurekaLog 421
- Reconfiguring EurekaLog for manual control 451

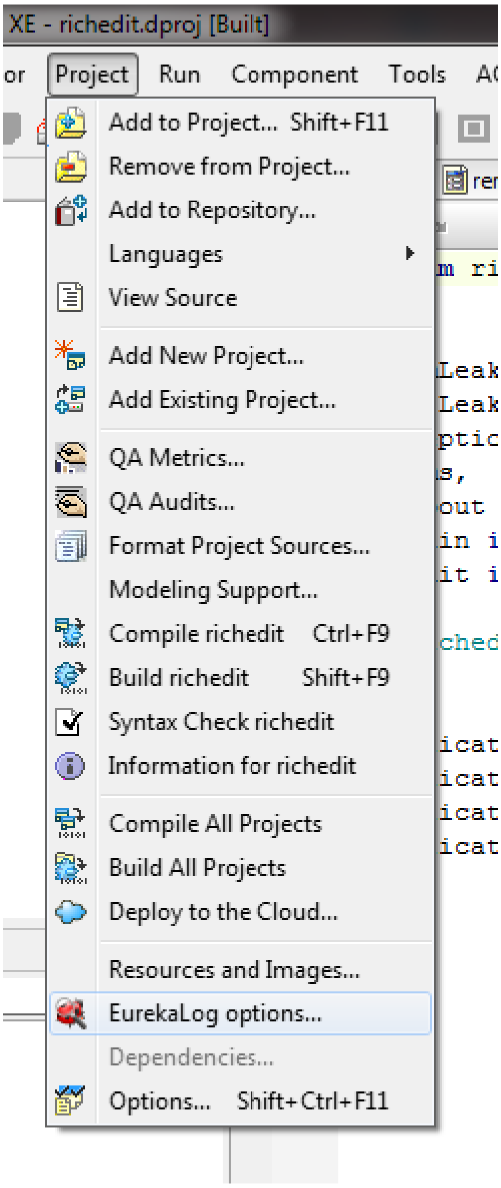
10.2 Interface

EurekaLog windows:
- IDE menu options 222:
  - EurekaLog project options 225
  - EurekaLog global options 230

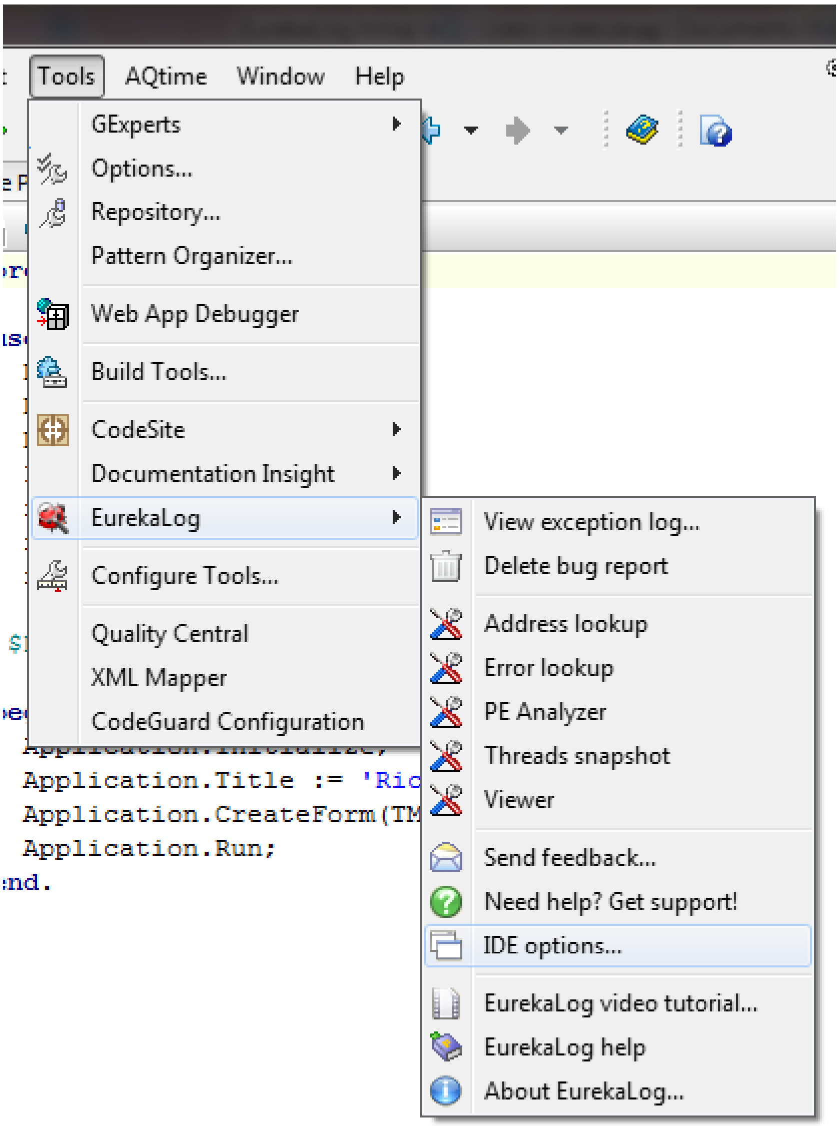
See also:
- Tools 617

10.2.1 IDE menu items (IDE commands)

EurekaLog places new menu items in IDE under **Project** and **Tools** root menu items:



EurekaLog menu items in Project menu



```
  Application.Title := 'Ric
  Application.CreateForm(TM
  Application.Run;
end.
```

EurekaLog menu items in Tools menu

Please note that exact location and items may be different - it depends on your environment (installed IDE experts).

Don't see those menu items? Check if EurekaLog is properly installed 604. Don't need these items? See alternatives 221.

There can be the following IDE menu items:

- **Project / EurekaLog options** - this command opens EurekaLog's settings for current project 225. If there is no current project, this command opens options for default project. This command should not be confused with **Tools / EurekaLog / EurekaLog IDE options** command (see below).
- **Tools / EurekaLog / View exception log** - this command opens EurekaLog Viewer with opened bug report for current project. If such bug report doesn't exists - no action taken (there will be informational message).
- **Tools / EurekaLog / IDE options** - opens EurekaLog settings 230. This is global settings for EurekaLog. It's different from **Project / EurekaLog options** command, which opens settings for current project only. I.e. **Project / EurekaLog options** command invokes global settings, **Tools / EurekaLog / EurekaLog IDE options** command invokes per-project settings.
- **Tools / EurekaLog / Address lookup** - this command is analog of Start menu's shortcut for Address Lookup tool 617.

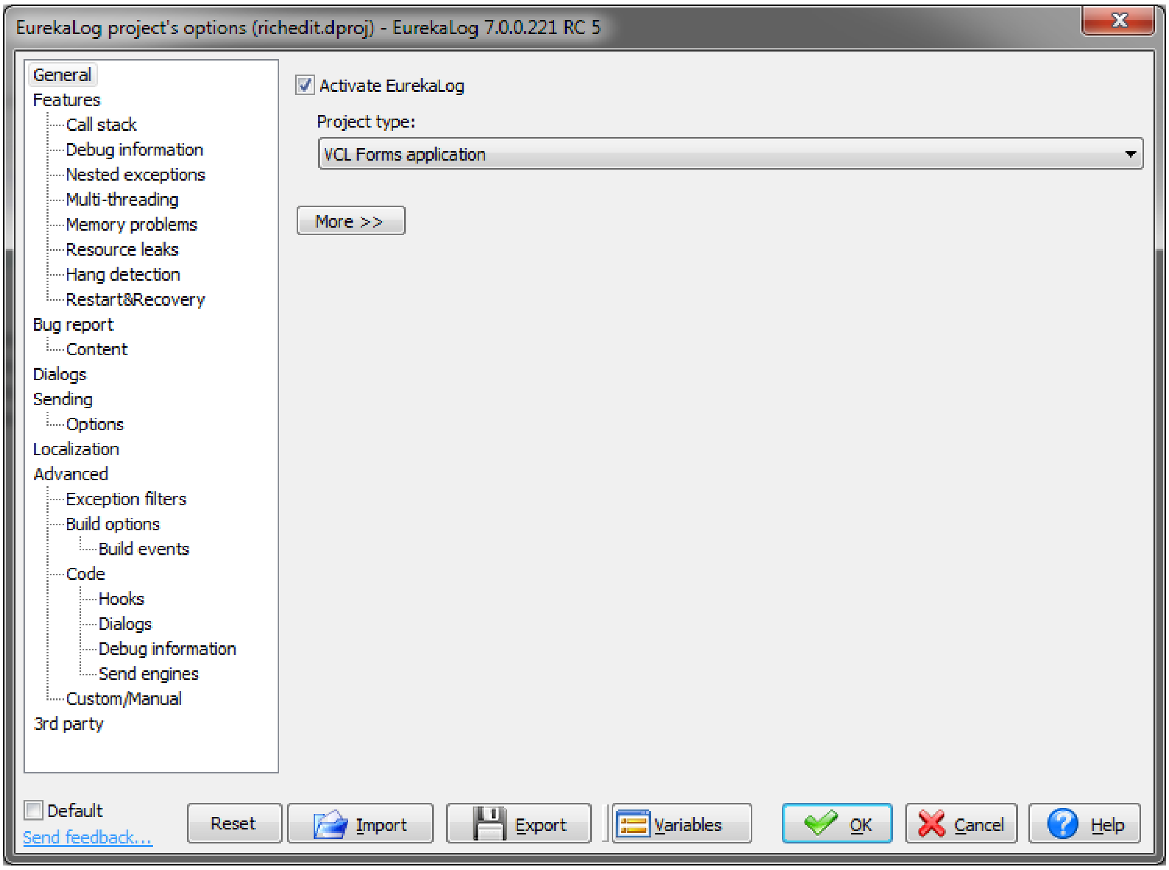
- **Tools / EurekaLog / Error lookup** - this command is analog of Start menu's shortcut for Error lookup tool 617.
- **Tools / EurekaLog / PE Analyzer** - this command is analog of Start menu's shortcut for PE Analyzer tool 617.
- **Tools / EurekaLog / Threads snapshot** - this command is analog of Start menu's shortcut for Thread Snapshot tool 617.
- **Tools / EurekaLog / Viewer** - this command is analog of Start menu's shortcut for EurekaLog Viewer tool.
- **Tools / EurekaLog / Send feedback** - this command opens your default e-mail client to allow you to share your feedback, opinions, make a suggestion or report a bug. Please, use the next command (**Need help? Get support!**) for asking questions.
- **Tools / EurekaLog / Need help? Get Support!** - this command opens your default internet browser to allow you to ask a question to our tech-support.
- **Tools / EurekaLog / EurekaLog Video Tutorial** - launches a video tutorial for new users.
- **Tools / EurekaLog / EurekaLog Help** - opens EurekaLog help file.
- **Tools / EurekaLog / About EurekaLog** - shows standard "About" window. You can see a version of installed EurekaLog here.

See also:

- EurekaLog tools 617

10.2.2 Project options

This is EurekaLog project's options dialog (do not confuse **EurekaLog project options** dialog with EurekaLog IDE options 230 dialog.):



EurekaLog project's options (richedit.dproj) - EurekaLog 7.0.0.221 RC 5

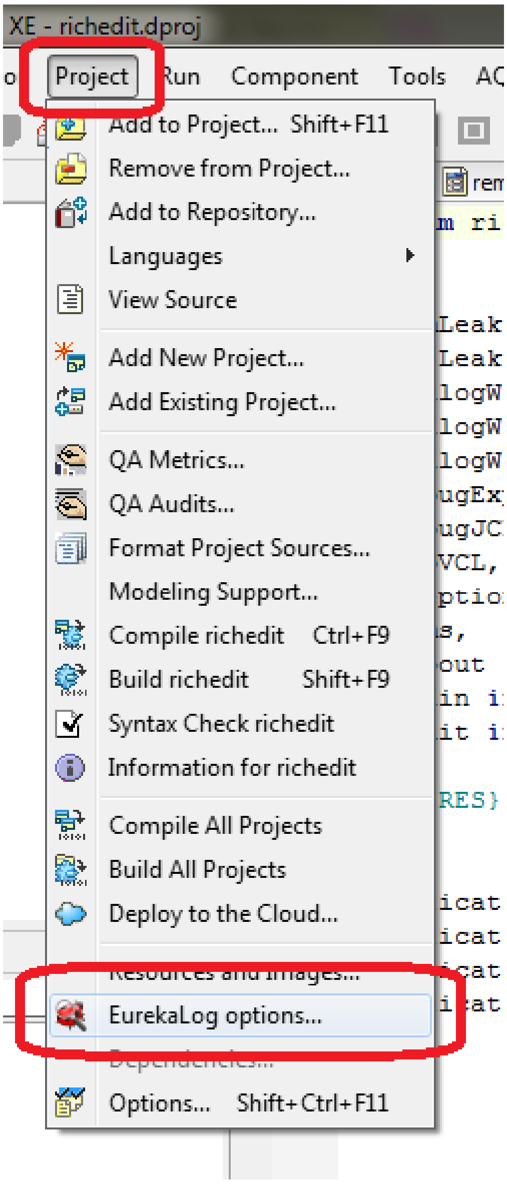
☑ Activate EurekaLog

Project type: VCL Forms application

□ Default

EurekaLog project options

You can open it via **Project / EurekaLog options** command (see also 222):



Opening EurekaLog project options

[Don't see EurekaLog menu items?](596)

[Don't use EurekaLog IDE expert?](221)

Please save your project before invoking this command.

This dialog allows you to edit project options, which are specific for EurekaLog. I.e. it allows you to edit EurekaLog options per project. There also another options dialog, which allows you to set global / IDE options for EurekaLog - it's located under **Tools** / **EurekaLog** / **IDE options** ([see also](230)).

The project options dialog works like any other dialog with OK and Cancel buttons. These options are stored in your project configuration - so be sure not to delete it. For example, Delphi 7 stores project configuration in .dof file. Delphi 2006 stores configuration in .bdsproj file. Delphi XE stores configuration in .dproj file.

If you don't want to store EurekaLog's options in project configuration - you can [export them to external .eof file](227) and use an [external settings editor](617) to edit them ([see also](421)).

This dialog have 3 parts:
- Buttons panel (bottom part)
- Menu (left tree)
- Options tabs (central part)

Bottom part with buttons is used for usual actions like OK, Cancel and Help (note: you can also press an F1 button). There are also buttons for:
- Importing and exporting project options 227
- Using variables 228
- Editing default project properties 230

Central part displays an options category, which is selected in the left menu. Therefore its content is different for different selected categories 234.

See also:
- EurekaLog project options 234
- Using EurekaLog options for customizing EurekaLog 180
- Working with EurekaLog configuration 439

10.2.2.1 Import / export settings

This is discussion of EurekaLog project's options 225.

You can export EurekaLog to external file and import settings from such files. EurekaLog uses ".eof" file extension to store external configuration. .eof-files are just renames text ini files 440.

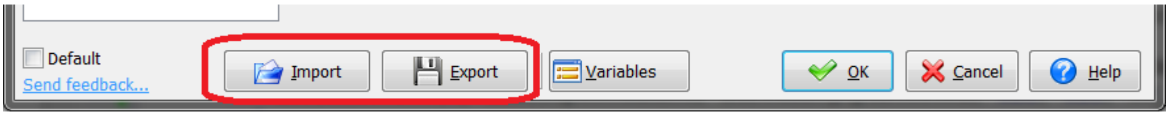
Options' exporting and importing are useful in the following cases:
- You want to transfer settings from one project to another. In this case you can open first project, export settings to file, then open second project and import options from file. Delete external (temporary) .eof-file after that.
- You want to store EurekaLog configuration outside of project's file (for example - to share same configuration between several projects). See this article 443 for more info.
- You want to specify alternative configuration to EurekaLog's post-processor 423. See also: command-line options 432 (you're interested in **--el_config** option).

.eof-files also can be edited outside of IDE either in text-editor or EurekaLog standalone Settings Editor tool.

Note: you may want to use common configuration instead of copying settings (import/export) between multiple projects. See also: using external configuration 443.

Open EurekaLog's project's settings 225.

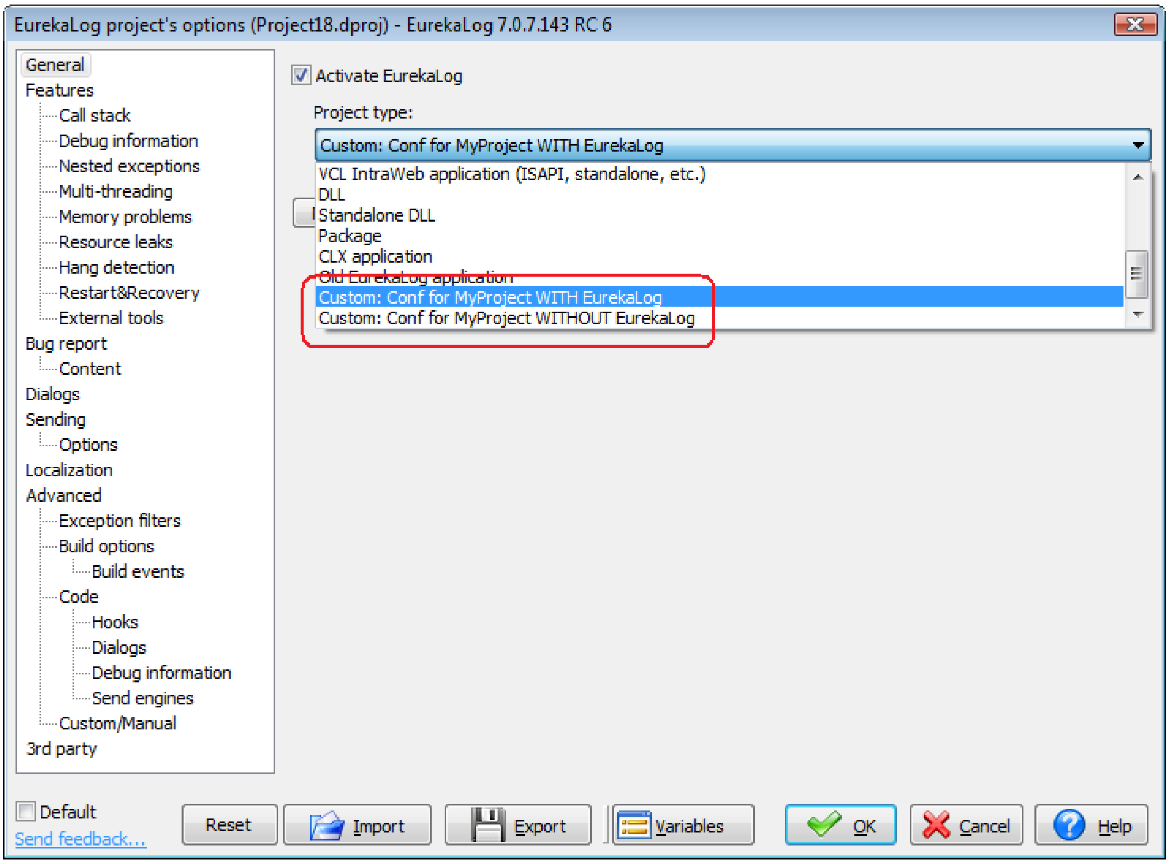
You can find "Import" and "Export" buttons at the dialog's bottom:



"Import" and "Export" buttons in EurekaLog project options dialog

Click on "Export" button to save current project's options to (new) external .eof-file.
Click on "Import" button to replace all current options with options from (existing) external .eof-file.

Default folder for saving .eof files (i.e. folder for "Import configuration" and "Export configuration" dialogs) is `%AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\` (e.g. like `C:\Users\`*`UserName`*`\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Profiles\`). Any .eof file placed in that location will appear as "custom" project type 363.



☑ Activate EurekaLog

Project type: Custom: Conf for MyProject WITH EurekaLog

VCL IntraWeb application (ISAPI, standalone, etc.)
DLL
Standalone DLL
Package
CLX application
Old EurekaLog application
Custom: Conf for MyProject WITH EurekaLog
Custom: Conf for MyProject WITHOUT EurekaLog

"Project type" option shows two .eof files

.eof files outside of the above mentioned folder will not appear in "Project type" option.

Apart from `%AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\` folder, a typical places to store .eof files are folder or sub-folder of your project.

See also:

- Storing EurekaLog options 440
- EurekaLog project's options 225
- Compiling your project with EurekaLog 421
- Using external configuration 443
- Working with EurekaLog configuration 439

10.2.2.2 Using variables

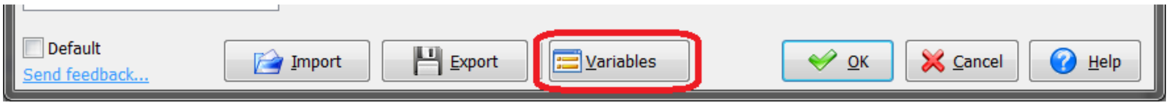
This is discussion of EurekaLog project's options 225.

You can use environment variables 413 in any text values in your project settings. You can insert variable by using "Variables" window. Variable is inserted as special tag. When you run your application at run-time, any variable value will be replaced with actual value, which is calculated at run-time.

For example, if you set your folder for saving bug-report to "%APPDATA%\MyBugReports" then your bug reports will be saved to `C:\Users\UserName\AppData\Roaming\MyBugReports\` or `C:\Documents and Settings\UserName\Application Data\MyBugReports` - depending on real value of %APPDATA% variable at run-time.

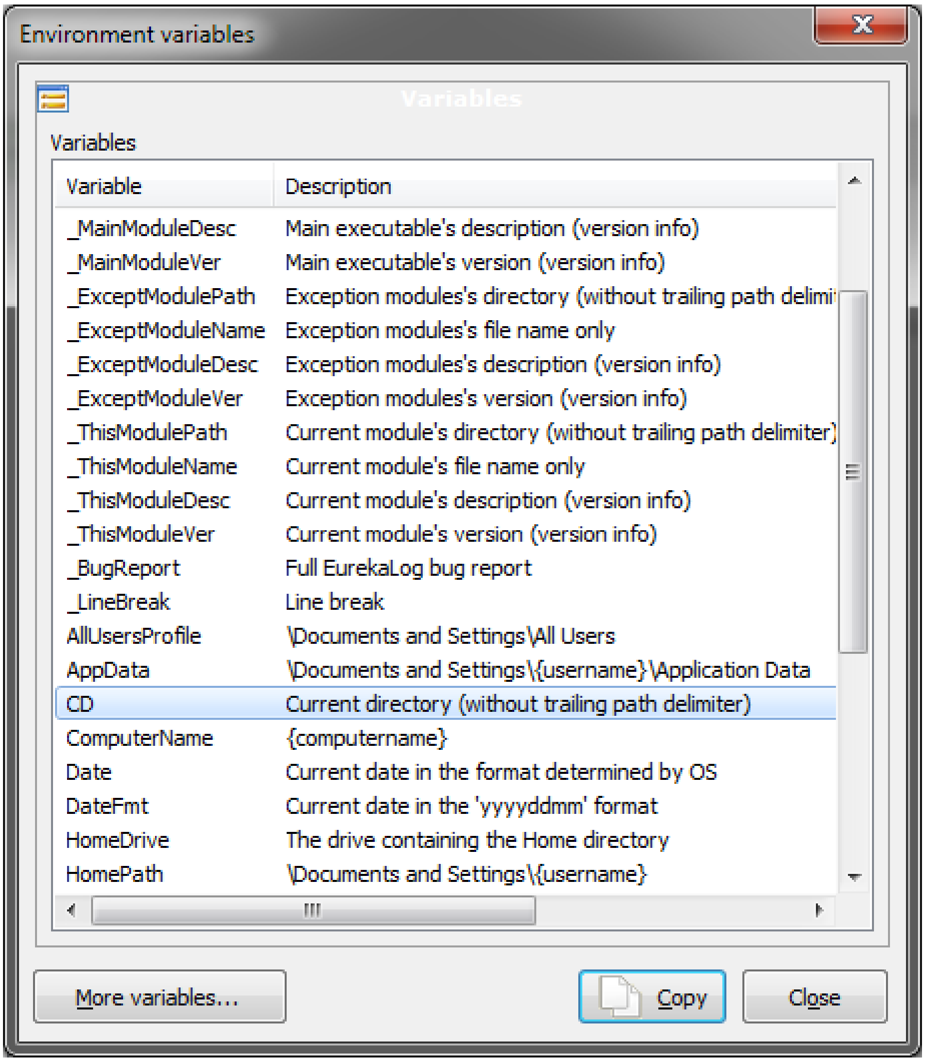
Note: variables names are case-insensitive.

When you're in your EurekaLog project options 225, you can click on "Variables" button at dialog's bottom:



"Variables" button in EurekaLog project options

Click on this button and you will see such window:



Variable	Description
_MainModuleDesc	Main executable's description (version info)
_MainModuleVer	Main executable's version (version info)
_ExceptModulePath	Exception modules's directory (without trailing path delimi
_ExceptModuleName	Exception modules's file name only
_ExceptModuleDesc	Exception modules's description (version info)
_ExceptModuleVer	Exception modules's version (version info)
_ThisModulePath	Current module's directory (without trailing path delimiter)
_ThisModuleName	Current module's file name only
_ThisModuleDesc	Current module's description (version info)
_ThisModuleVer	Current module's version (version info)
_BugReport	Full EurekaLog bug report
_LineBreak	Line break
AllUsersProfile	\Documents and Settings\All Users
AppData	\Documents and Settings\{username}\Application Data
CD	Current directory (without trailing path delimiter)
ComputerName	{computername}
Date	Current date in the format determined by OS
DateFmt	Current date in the 'yyyyddmm' format
HomeDrive	The drive containing the Home directory
HomePath	\Documents and Settings\{username}

"Variables" dialog

"Copy" button will close the window with copying selected variable into clipboard. Alternatively: you can just double-click on variable in the list.
"Close" button will close the window without any action.

Suggested action's sequence:

- Open "Variables" window.
- Select variable that you want to use (see Variables 413 for description).
- Click "Copy" (or double-click variable). Dialog will be closed.
- Paste variable name from clipboard (`Ctrl + V` or `Shift + Ins`) to target setting's edit box.

Note: this dialog suggests you only build-in special pseudo-variables (those with names started with "_") and commonly used variables. However, you can use any environment variable (even if it's not listed in this dialog). See also: environment variables 413.

See also:

- Variables 413
- EurekaLog project's options 225

10.2.2.3 Changing default properties

This is discussion of EurekaLog project's options 225.

Important Note: this feature exists for backward compatibility. We do not recommend to use it. Consider using custom profiles instead 227.

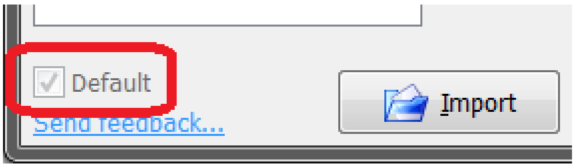
Default project settings is the settings which are applied automatically to each new created project (only if EurekaLog IDE expert 221 is installed).

EurekaLog's configuration of default project is stored in the same location as IDE's configuration of default project, but in another file: file extension is changed to .eof. For example:

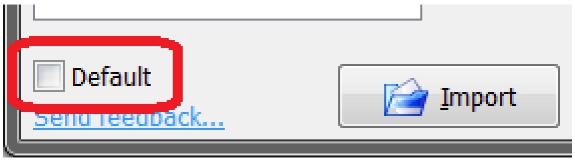
- `C:\Users\User\AppData\Roaming\Embarcadero\BDS\8.0\DefProject.eof` - Delphi XE (IDE default project configuration is stored in DefProject.bdsproj)
- `C:\Program Files\Borland\Delphi 7\Bin\DefProj.eof` - Delphi 7 (IDE default project configuration is stored in DefProj.dof)

You can edit this configuration either by opening it in the text editor or standalone EurekaLog configuration editor, or directly from IDE.

To edit default project configuration: open EurekaLog's project options 225 and check the "Default" checkbox at the dialog's bottom-left corner. This checkbox will be checked by default if there is no project. For example:



"Default" checkbox is checked and disabled, if there is no active project



"Default" checkbox is unchecked and enabled, if you're editing project's options

Edit project's settings as you like, check the "Default" checkbox and close dialog with the "OK" button. Current settings will be saved as defaults.

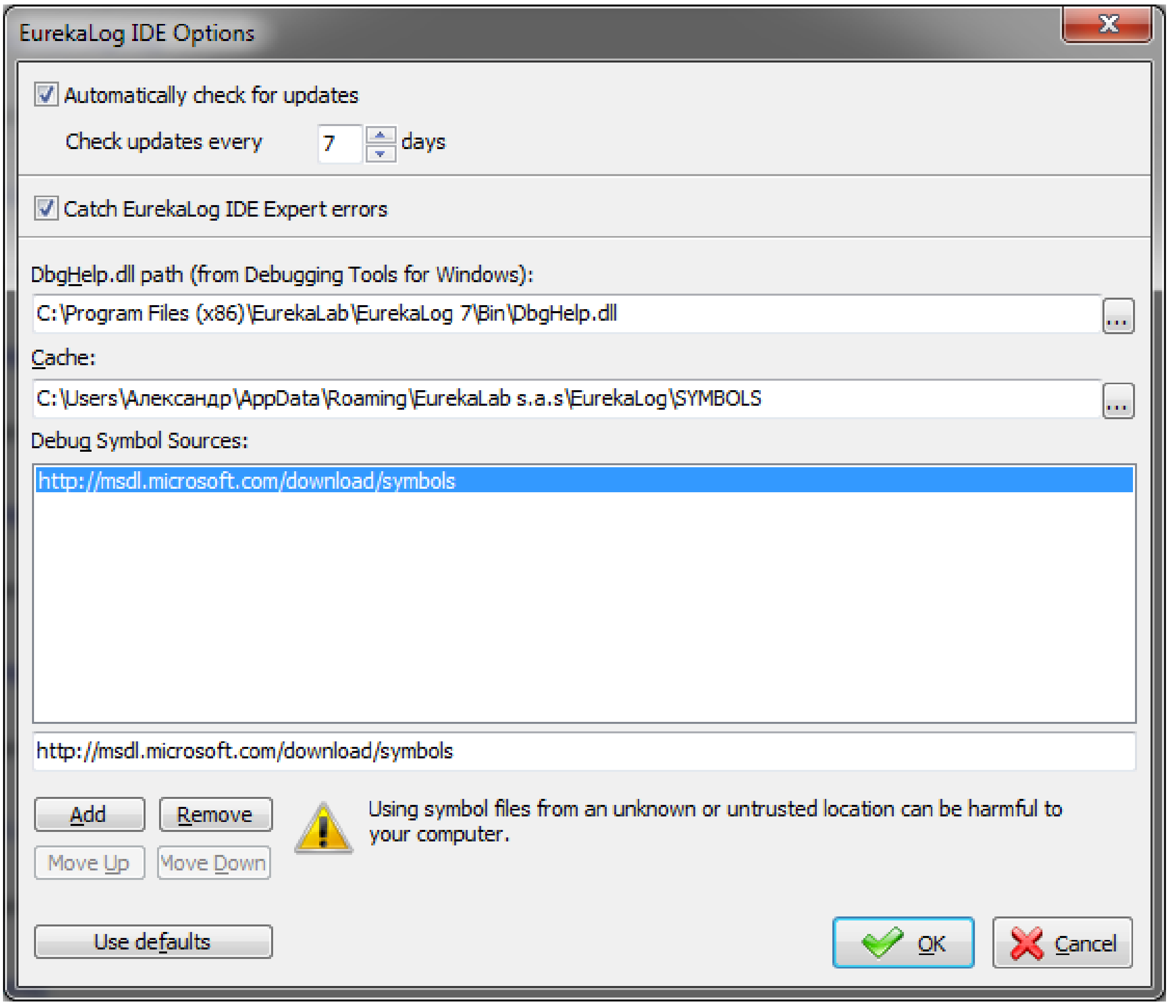
Note: if you have an active project opened - its settings will be changed as well.

See also:

- EurekaLog project's options 225

10.2.3 IDE options

This is EurekaLog IDE options dialog (do not confuse **EurekaLog IDE options** dialog with EurekaLog project options 225 dialog.):



☑ Automatically check for updates

Check updates every 7 days

☑ Catch EurekaLog IDE Expert errors

DbgHelp.dll path (from Debugging Tools for Windows):

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Bin\DbgHelp.dll

Cache:

C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\SYMBOLS

Debug Symbol Sources:

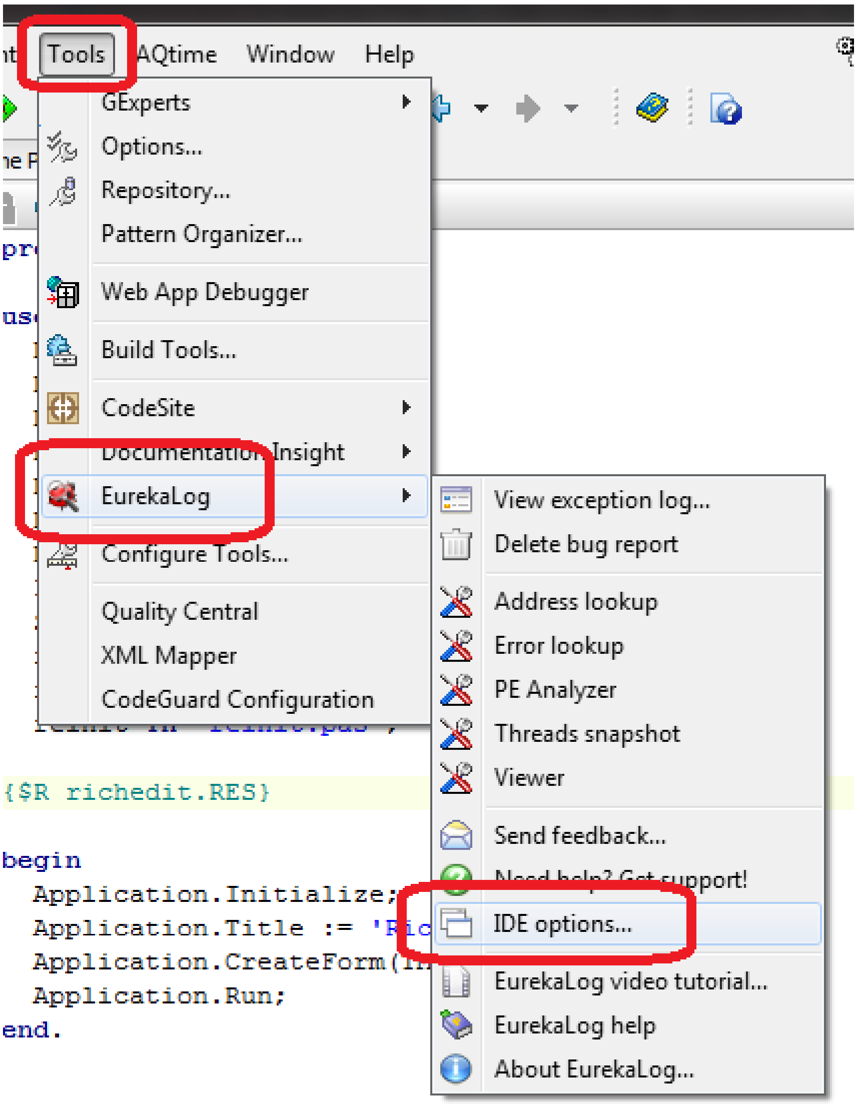
http://msdl.microsoft.com/download/symbols

http://msdl.microsoft.com/download/symbols

Using symbol files from an unknown or untrusted location can be harmful to your computer.

EurekaLog IDE options

You can open it via **Tools / EurekaLog / IDE options** command (see also 222):



```
{$R richedit.RES}

begin
  Application.Initialize;
  Application.Title := 'Ric
  Application.CreateForm(Ti
  Application.Run;
end.
```

Opening EurekaLog IDE options

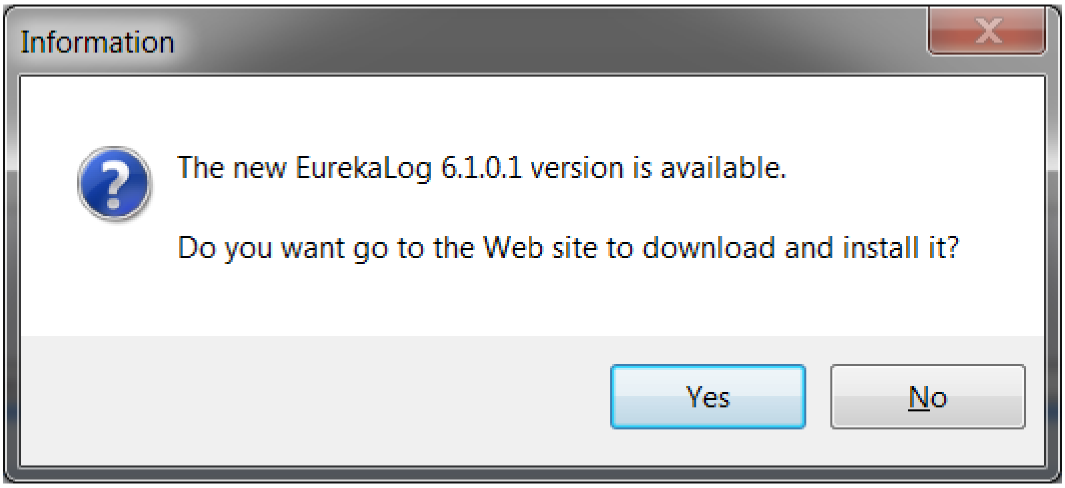
Don't see EurekaLog menu items? 596

Don't use EurekaLog IDE expert? 221

This dialog allows you to edit global EurekaLog options. There is also another options dialog, which allows you to set per-project settings for EurekaLog - it's located under **Project / EurekaLog options** (see also 225).

The IDE options dialog works like any other dialog with OK and Cancel buttons. These options are stored in the registry (under HKEY_CURRENT_USER).

1. "**Automatically check for updates**" option will check for new version of EurekaLog available once in specified number of days. We recommend you to keep this option on and update EurekaLog when new version will be available. When new version of EurekaLog is found, you will be prompted with such dialog at IDE's startup:


Information

The new EurekaLog 6.1.0.1 version is available.

Do you want go to the Web site to download and install it?

Found a new EurekaLog's version

"Yes" button will open your default browser on EurekaLog's site, where you can get a new version of EurekaLog. You can continue to work in IDE.
"No" button will just continue work without opening browser.

2. "**Catch EurekaLog IDE Expert errors**" option installs hooks in IDE to catch problems within EurekaLog IDE expert. When this option is enabled: each unhandled exception for EurekaLog IDE expert will generate bug report which can be send to developers (us). Exceptions for IDE itself and for other extensions are not changed. When this option is disabled: any unhandled exception for EurekaLog IDE expert will generate a simple error dialog only (message box) without bug report and additional info.

Enable this option to create bug reports about problems in EurekaLog (you can send these reports to us).
Disable this option to improve EurekaLog IDE expert speed (only a little) and to improve compatibility with other 3rd party software.

Note: you need to restart IDE for this option to take effect.

3. "**DbgHelp.dll path**" option specifies full file name to DbgHelp DLL. You can find it in EurekaLog's installation folder (like: `C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\dbghelp.dll`). You can also change this path to alternative library. This is DLL from Debugging tools for Microsoft Windows. It's used in MS Debug stack tracing method and MS Debug info provider.

4. "**Cache**" option specifies read-write folder to be used as cache for debug info symbols. It's empty by default, which means disabled feature. You can click on "**Use defaults**" button to set default preference (which is sub-folder in your %APPDATA%, for example: `C:\Users\User\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\SYMBOLS`) or select your own folder. Be sure that this folder is writable and disk have some free space (up to 500 Mb). This cache folder is used by MS Debug info provider to store debug symbols for system libraries.

5. "**Debug symbol sources**" option specifies debug info sources for MS Debug info provider. You can add one or more sources here by using edit-box and buttons below. Source can be local folder (like: C:\Symbols), shared network path (like: \\server\symbols) or URL of symbol's server (like: http://server/symbols). It's empty by default, which means disabled feature. You can click on "**Use defaults**" button to set default preference (which is default Microsoft's symbol server: `http://msdl.microsoft.com/download/symbols`) or select your own sources.

See also:
- MS Debug info provider 412
- Enabling debug information for Windows DLLs 504

10.3 Options

There are the following categories available:

- General 234
- Features 237
- Bug report 264
- Dialogs 267
- Sending 302
- Localization 339
- Advanced 341
- 3rd party 357

Note: description of almost every option contains name of corresponding property of `TEurekaModuleOptions` class. For example:

1. "**Save bug report to file**" (`.SaveLogFile`) option enables saving...

This means that you can change this option at run-time by altering `.SaveLogFile` property of either global `CurrentEurekaLogOptions` or local `Options` property of exception object in event handlers.

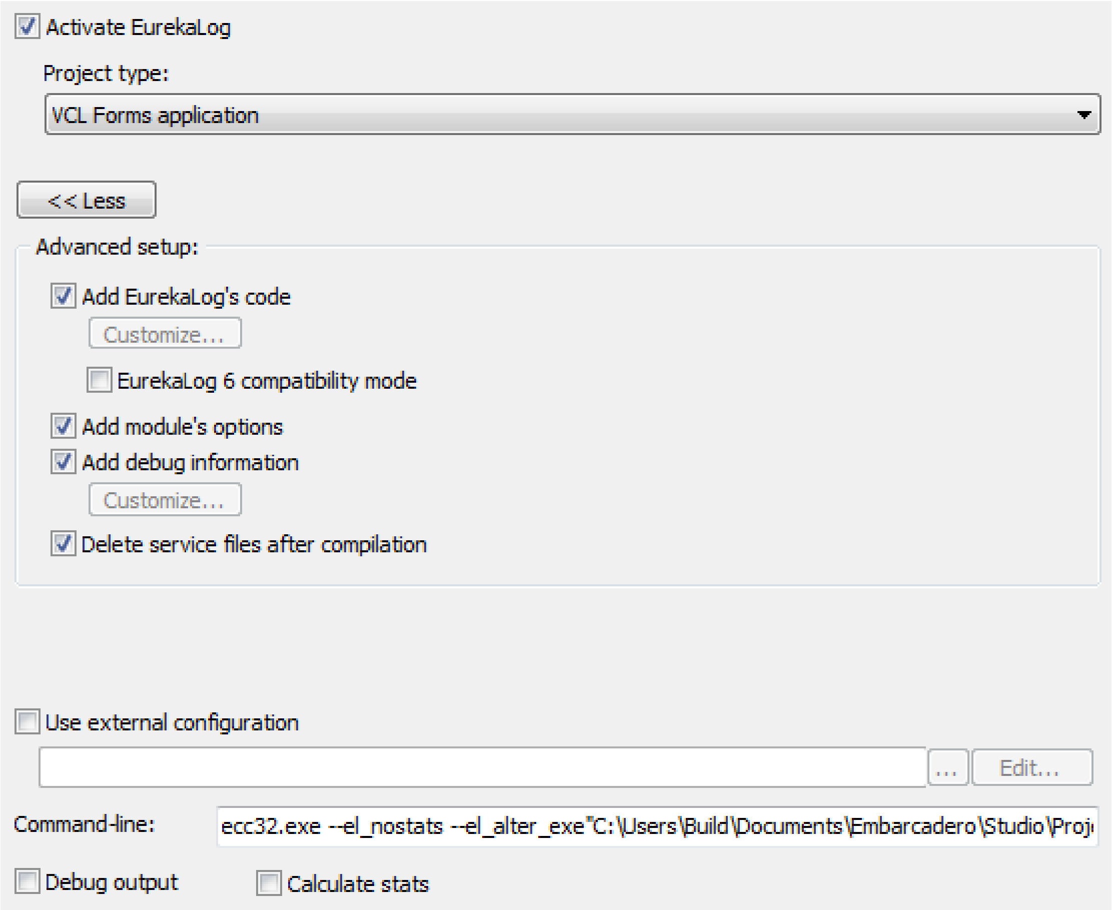
If there is no mention of corresponding property - this means that either this option can not be changed at run-time (e.g. option affects only compilation - for example, stripping relocation tables, or encrypting injecting debug information; or option can not be changed after initial setup - for example, using low-level hooks, or installing memory manager filter to catch leaks), or option is changed by other means (for example, leaks and memory options are changed by routines in `EMemLeaks` unit; hang detection options are changed by routines in `EFreeze` unit).

See also:

- IDE EurekaLog project options dialog 225
- Import / Export settings 227
- Using variables 228
- Changing default properties 230

10.3.1 General page

This is "General" page in EurekaLog project's options 225.



☑ Activate EurekaLog

Project type: VCL Forms application

Advanced setup:

☑ Add EurekaLog's code

□ EurekaLog 6 compatibility mode

☑ Add module's options

☑ Add debug information

☑ Delete service files after compilation

□ Use external configuration

Command-line: ecc32.exe --el_nostats --el_alter_exe"C:\Users\Build\Documents\Embarcadero\Studio\Proj

□ Debug output □ Calculate stats

General options

General page contains EurekaLog activation options. If you need to enable/disable EurekaLog in your application - you can do it here.

1. "**Activate EurekaLog**" option enables EurekaLog in your project. Check this checkbox to enable EurekaLog in your application and uncheck it to disable EurekaLog for your application. See also: enabling EurekaLog in your application 33 and Compiling your project with and without EurekaLog 445.

This checkbox sets status of EurekaLog (active or not). It acts as "meta-switch" for checkboxes in "Advanced setup" group box. Usually, you don't need to set advanced options manually: just check/uncheck "Activate EurekaLog" checkbox and set your application type 363 - that's all.

Change checkboxes in "Advanced setup" only when you need a custom/non-standard behaviour.

Note: switching "Activate EurekaLog" option *may* be not an appropriate way to compile your project with and without EurekaLog 445.

2. "**Project type**" option selects type of your application 363. See also: enabling EurekaLog in your application 33.

Note: this option will load all custom saved external configurations, which were saved to default folder on export (via "Export" button). See Working with configurations 439 and Compiling your project with and without EurekaLog 445 for more information. Custom configurations will be listed at the end of the list.

Advanced options - use for custom setups:

3. "**Add EurekaLog's code**" option includes EurekaLog's units in your application. Click on "Customize" button to select which units include in your application. You can uncheck this checkbox, if you don't want to include EurekaLog's code in current executable.

4. "**Add module's options**" option adds EurekaLog project options to your application. You set EurekaLog project options in this dialog 225. Including EurekaLog's options is necessary for EurekaLog to function. You can uncheck this checkbox, if you don't want to include EurekaLog's code in current executable.

Note: this option must be set if you check either "Add EurekaLog's code" or "Add debug information" options.

5. "**Add debug information**" option injects debug information 40 in EurekaLog's format into your application. Debug information is necessary to build a human-readable call-stack and get textual descriptions of code's locations. You can uncheck this checkbox if you plan to use debug information in other format.

6. "**Delete service files after compilation**" option allows you to clean up your project from unnecessary files. EurekaLog asks linker to generate .map/.tds/.drc files to read debug information from them. Once debug information is injected - these files are no longer needed. This option can be used to delete them automatically. See EurekaLog's basics 38 for more information. You can turn this option off to keep files untouched (if you want them for other 3rd party software). See also: External tools options page 259.

Important Note: enabling this option may prevent other tools from functioning. Specifically, enabling this option prevents source-code debugging in C++ Builder 614.

7. "**Use external configuration**" option allows you to use EurekaLog configuration in external .eof file instead of configuration inside project. Checking this option will disable all other options in this dialog 225. You also need to specify .eof file to use. You can use "..." button to select file and "Edit" button to edit settings in external file (the last action requires file association with .eof files). See also: using external configuration 443. You can use %_IDESrc% environment variable 413 to specify locations relative to your project's folder.

8. "**Command-line**" option is read-only. It's for informational purposes only. This edit box shows command-line to run to enable EurekaLog in your project after compilation (if you're compiling outside of IDE). Please, see Compiling your project with EurekaLog 421 article for more information. This edit box will indicate "N\A" if there is no project opened.

This option is affected by "Use external configuration", "Debug output" and "Calculate stats" options.

Note: you don't need to run this command if you compile your project in IDE with EurekaLog IDE expert 221 installed.

9. "**Debug output**" option will enable detailed debug mode for output from EurekaLog command-line compiler 423. Detailed output will appear in IDE's Messages window ("Build" tab).

Use this option for diagnostic purposes. Disable for normal daily usage.

10. "**Calculate stats**" option will collect statistics about EurekaLog's work and your compiled application. You have to rebuild the project to collect statistics. Collected stats will appear in IDE's Messages window ("Build" tab). You can also view them at "Statistics" tab 357. Build stats are not saved when project is closed.

Note: enabling both "Calculate stats" and "Debug output" options will greatly increase amount of collected statistics.

See also:

- EurekaLog's basics 38

- Enabling EurekaLog in your applications 33
- Basic procedures 45
- Application types 363
- Typical scenarios 68
- Migration guide 624
- Compiling your project with EurekaLog 421
- Working with EurekaLog configuration 439
- External tools options page 259

10.3.2 Features page

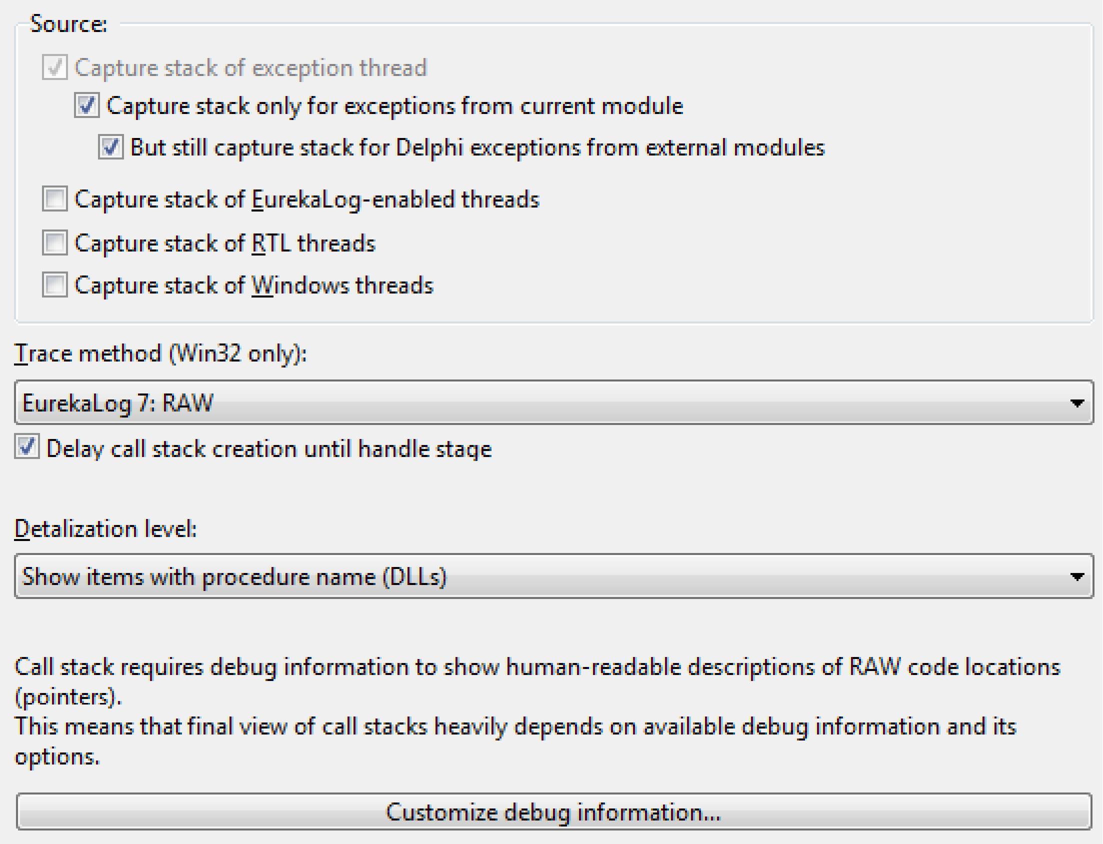
This is root category for core and additional features of EurekaLog.

These features are:

- Call Stack 237
- Debug information 243
- Nested/Chained exceptions 244
- Multi-threading 246
- Memory problems 250
- Resource leaks 255
- Hang detection 257
- Restart&Recovery 259
- External tools 259

10.3.2.1 Call Stack page

This is "Call Stack" page in EurekaLog project's options 225.


Source:

☑ Capture stack of exception thread

☑ Capture stack only for exceptions from current module

☑ But still capture stack for Delphi exceptions from external modules

□ Capture stack of EurekaLog-enabled threads

□ Capture stack of RTL threads

□ Capture stack of Windows threads

Trace method (Win32 only): EurekaLog 7: RAW

☑ Delay call stack creation until handle stage

Detalization level: Show items with procedure name (DLLs)

Call stack requires debug information to show human-readable descriptions of RAW code locations (pointers).
This means that final view of call stacks heavily depends on available debug information and its options.

Call Stack options

Options on "Call Stack" page allow you to customize EurekaLog behavior related to call stacks in bug reports.

1. **"Capture stack only for exceptions from current**

module" (.csoCaptureOnlyModuleExceptions) option allows you to speed up execution by ignoring all exceptions outside of your executable module.

Since normal practice for exceptions is to handle them within the same module 457 - exceptions usually do not leave module (i.e. they are not shared between modules). This means that you're usually interested only in exceptions from the same module. This option allows you to ignore any other exception.

Note: this option is extremely useful in applications with plug-ins (including COM modules).

It's recommended to keep this option checked when possible. Disable this option for packaged applications or other application types which includes sharing exceptions between modules. However, consider using (checking) "**But still capture stack for Delphi exceptions from external modules**" option instead of unchecking this option.

2. "**But still capture stack for Delphi exceptions from external modules**" (.csoCaptureDelphiExceptions) option excludes Delphi exceptions from "**Capture stack only for exceptions from current module**" option.

Checked: Delphi exceptions are always captures - regardless of executable module which raises it
Unchecked: all exceptions from external modules are ignored

Note: this option has no effect if "**Capture stack only for exceptions from current module**" option is not checked.

3. "**Capture stack of EurekaLog-enabled threads**" (.csoShowELThreads) option includes call stacks of all EurekaLog-enabled threads in application - regardless of thread type. By default only exception thread is captured.

"EurekaLog-enabled thread" term refers to thread with enabled per-thread EurekaLog. You can enable EurekaLog in any thread by calling SetEurekaLogStateInThread function 570 or just simply create threads with TThreadEx 559 or BeginThreadEx 551. See Enabling EurekaLog for background threads 568 for more details.

Turn this option off for single-threaded application.
Turn this option on for multi-threaded application.

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will also require more time during exception processing.

4. "**Capture stack of RTL threads**" (.csoShowRTLThreads) option includes call stacks of all RTL threads in application. By default only exception thread is captured.

"RTL threads" means threads started with TThread or BeginThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread together with "Capture stack of EurekaLog-enabled threads" option instead.
Turn this option on to capture call stack of external RTL threads (that is threads started by 3rd party code without your control).

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will require more time during exception processing.

5. "**Capture stack of Windows threads**" (`.csoShowWindowsThreads`) option includes call stacks of all non-RTL threads in application. By default only exception thread is captured.

"Windows threads" means threads started with CreateThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread together with "Capture stack of EurekaLog-enabled threads" option instead. Alternatively, you may use "Capture stack of RTL threads" option instead.
Turn this option on to capture call stack of external Windows threads (that is threads started by 3rd party code without your control).
Never start your own thread with CreateThread function.

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will require more time during exception processing.

6. "**Trace method**" (`.csoAllowedRenderMethods`) option selects method for creating/building call stack. This option is applicable to x86-32 only, it has no effect for x86-64 platform.

The following methods are supported:

- x86-32:
  - EurekaLog 7: stack frames
  - EurekaLog 7: RAW (**recommended**)
  - Memory debugger: stack frames (fast)
  - Memory debugger: RAW
  - Compatibility: EurekaLog 6
  - Compatibility: JCL (stack frames)
  - Compatibility: JCL (RAW)
  - Compatibility: Microsoft (requires external DLL)
  - Compatibility: madExcept (requires madExcept installed)
- x86-64:
  - OS tracing

6.1. "EurekaLog 7" methods are default for EurekaLog 7. This is recommended choice for new projects. This is new and improved EurekaLog stack tracing method. Improvements are done for filtering out false-positive calls. Chose these methods to get more adjusted call stack. Chose EurekaLog 6 method (see item 4.3 below) to get unfiltered call stack.

There are RAW and stack frame methods. See this explanation 578 for differences between stack frames and RAW. RAW method includes all items from stack frames method and adds more.

6.2. "Memory debugger" methods are lightweight and fast methods which are used by memory and resource profiling features of EurekaLog. It has to be as fast as possible, so it lacks advanced checks and may introduce many inaccurate and false-positive entries.

See this explanation 578 for differences between stack frames and RAW.

6.3. "Compatibility" methods are methods for compatibility with legacy and 3rd party code. It's not recommended to use these methods in new projects. Use these methods only for old code, which may expect certain call stack items.

Microsoft stack tracing method requires DbgHelp DLL. See: Using Microsoft's DbgHelp DLL 504
.

madExcept stack tracing method requires madExcept installed. You will need also to include `EStackTracingMadExcept` unit into your project file.

7. "**Delay call stack creation until handle stage**" option postpones analyzing exception for

later stages of processing (if possible).

Please, see this article 583 for detailed explanation of delayed (deferred) call stacks.

Enable this option for better performance.
Disable this option for better detalization and compatibility.

Notes:

- enabling some advanced features of EurekaLog (such as handling safecall exceptions, using exception filters with "Exception Kind" <> "All", etc.) may require creating call stack earlier than usual (for example: to detect if exception is raised within safecall wrapper), so this option will have no effect.
- carefully use this option for multi-threaded application. Deferred call stack creation means that call stacks of other (non-exception) threads will be captured much later - when original exception is handled/processed. Therefore, call stacks of other threads will not represent threads states when exception occurred.

8. "**Detalization level**" option indicate which items should be included in call stack:

- `.csoShowInvalid` - Show any (including RAW addresses)
- `.csoShowPointers` - Show any item belong to executable module (unknown locations within DLL)
- `.csoShowDLLs` - Show items with procedure name (DLLs)
- `.csoShowBPLs` - Show items with unit name (BPLs)
- Show only items with full info (line number available)

First list's item will not restrict call stack's item at all. Everything will be added. Each following list's item restricts added call stack's items. Last list's item is most restrictive: only locations with full debug information available will be added to call stack. See these articles for examples:

- Configuring call stack 48
- Using EurekaLog with DLL compiled with 3rd party compiler 496

Recommended value is "Show items with procedure name (DLLs)" or "Show items with unit name (BPLs)".

Important Note: "Show any (including RAW addresses)" value is not recommended to be used in typical application. This value is intended to be used with dynamically generated code. Such code has no debug information available and it does not belong to any executable module, but rather it is allocated in dynamic memory (heap) or CPU stack. Use "Show any item belong to executable module (unknown locations within DLL)" instead of "Show any (including RAW addresses)" for applications that do not generate code dynamically.

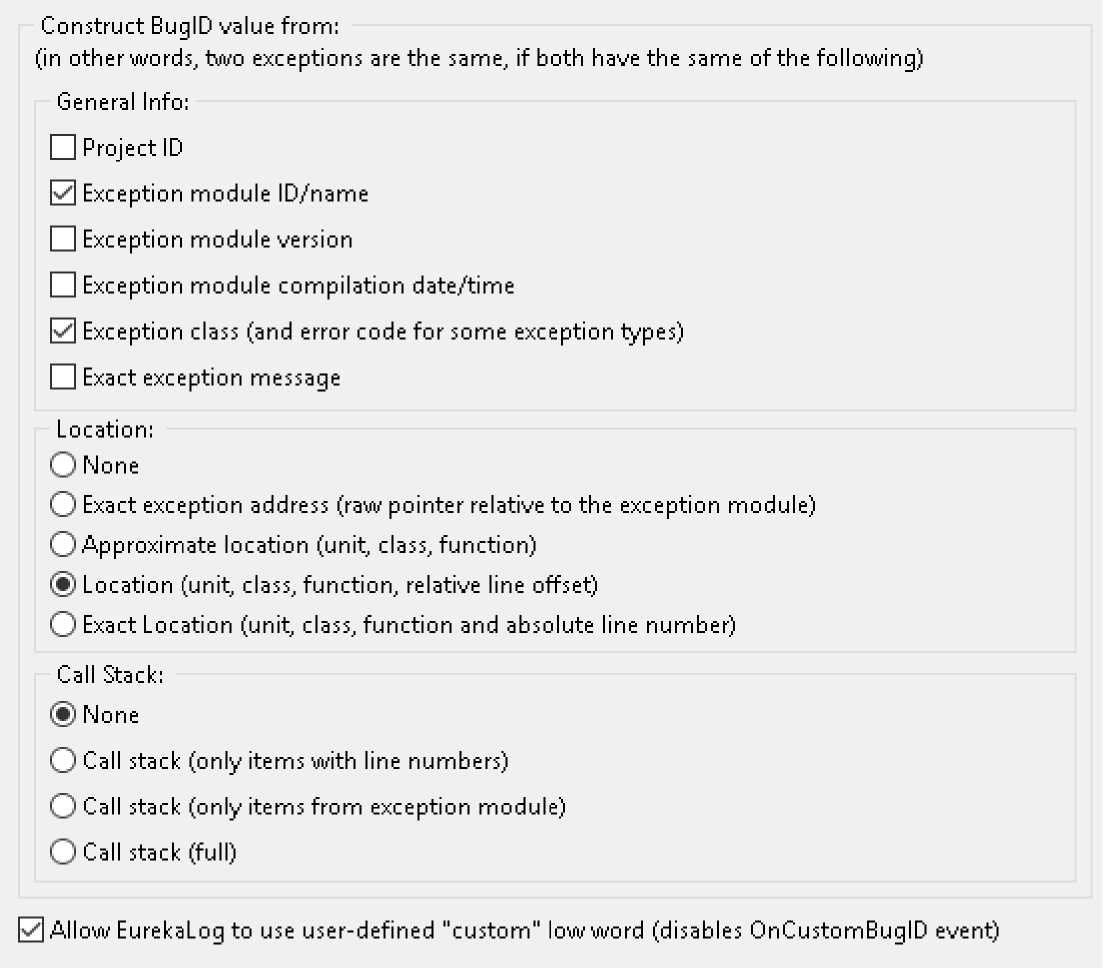
Note: "Show any (including RAW addresses)" and "Show any item belong to executable module (unknown locations within DLL)" values will disable speed optimizations 259.

See also

- Configuring call stack 48
- Customizing debug information 243
- RAW and stack frames methods 578
- Call stacks 79
- Using EurekaLog in multi-threaded applications 547
- Chained exceptions 573
- Using Microsoft's DbgHelp DLL 504
- Using EurekaLog in DLL 455
- Using EurekaLog with DLL compiled by 3rd party compiler (Microsoft Visual Studio, etc.) 496

10.3.2.2 BugID page

This is "Bug ID" page in EurekaLog project's options 225.



Construct BugID value from:
(in other words, two exceptions are the same, if both have the same of the following)

General Info:

☐ Project ID

☑ Exception module ID/name

☐ Exception module version

☐ Exception module compilation date/time

☑ Exception class (and error code for some exception types)

☐ Exact exception message

Location:

○ None

○ Exact exception address (raw pointer relative to the exception module)

○ Approximate location (unit, class, function)

● Location (unit, class, function, relative line offset)

○ Exact Location (unit, class, function and absolute line number)

Call Stack:

● None

○ Call stack (only items with line numbers)

○ Call stack (only items from exception module)

○ Call stack (full)

☑ Allow EurekaLog to use user-defined "custom" low word (disables OnCustomBugID event)

BugID options

These options allow you to customize BugID 421 generation without need to write your own event handlers. Each option on this page will include (when checked) or exclude (when unchecked) corresponding information from BugID generation. In other words, if some option is checked - then two exceptions will be considered to be the same if both have the same information (and exceptions will be considered to be different - if information is different). If some option is unchecked - then it will have no effect on comparison of exceptions.

Important Note: if all options on this page are disabled - then each exception will have an unique BugID.

1. "**Project ID**" option will include Project ID (PID). Project ID is a GUID value which is unique for each project. It is used by EurekaLog to identify projects. Different projects are supposed to have different project IDs.

By default - project ID is extracted from .dproj, .bdsproj, .cproj files and then stored in base configuration (.eof file). You may specify project ID manually by using ecc32 with --el_pid switch 432.

It is usually a good idea to enable this option when you are going to send reports to bug tracker or any other type of centralized storage. That way exceptions from one project will not be mixed with exceptions from other project.

2. "**Exception module ID/name**" option will include Module ID of executable module in which exception has occurred. Module ID is module name for non-EurekaLog modules, it is GUID (Project ID) for EurekaLog modules.

If you turn this option off - be sure you are using some other options to identify exception.

3. "**Exception module version**" option will include version number. Version information is taken from executable version info block.

If you enable this option - then same exception from different versions of your application will be considered as two different exceptions. This is useful option to be used with bug trackers or any other system which sorts bug reports by version.

4. "**Exception module compilation date/time**" option will include compilation date/time of exception's module.

This option is very similar to "**Exception module version**" option. Usually you should add a version information to your modules and use "**Exception module version**" option instead. Use "**Exception module compilation date/time**" option only when you are working with modules without version information.

5. "**Exception class (and error codes for some exception types)**" option will include name of exception object's class (such as 'EAccessViolation', etc.). Additionally, some exception types (such as EOSError or EDatabaseError, etc.) will include an error code or error message - to distinguish between different exceptions of the same class.

If you turn this option off - be sure you are using some other options to identify exception.

6. "**Exact exception message**" option will include text message from exception object.

Since exception message may include localized parts (such as messages from OS or your own localizations) as well as variable parts (such as file names in message text) - it is not a good idea to enable this option, because changes in localization, environment will make same exception to become two different exceptions. Enable this option only if receive merged reports for different exceptions and you have no other option. We strongly recommend to use OnCustomBugID event to provide custom BugID instead of using this option.

7. "**Location: Exact exception address**" option will include RAW offset (in bytes) from beginning of exception module.

If you enable this option - then same exception from different compilations of your application will be considered as two different exceptions (unless recompilation will produce exactly the same executable).

This option can be used with either "**Exception module version**" or "**Exception module compilation date/time**" (or both) options enabled, as well as replacement for both of these options.

8. "**Location: Approximate location**" option will include unit name, class name, function name, but will not include address, offset or any line number information. This option allows to identify exception even if source code is changed.

If you enable this option - be sure to also enable identification by exception class and/or message.

9. "**Location: Location**" option will include unit name, class name, function name and offset in lines from first line of the function. This allows to identify exception line even if source code before exception function is changed, but it will not identify exception if function itself was changed.

10. "**Location: Exact location**" option will include full information: unit name, class name, function name, and line number. This will allow to identify exception after any possible recompilation, but not after source code changes.

11. "**Call Stack: Only items with line numbers**" option will include only call stack items with line numbers present. This usually means Delphi source code only (even from other modules). Items will be included as: unit, class, function, line offset (e.g. same as with "**Location: Location**" option).

You may use this option to ignore calls to system DLLs or even RTL/VCL code (if you have "Use Debug DCUs" option disabled).

Note: first item (exception address) is ignored. Use "Location" options above to include/ exclude call stack's exception entry.

12. "**Call Stack: Only items from exception module**" option will include only call stack items from exception module. Items will be included as: unit, class, function, line offset (e.g. same as with "**Location: Location**" option).

You may use this option to include calls from RTL/VCL code if you have "Use Debug DCUs" option disabled, but ignore calls to system DLLs.

Note: first item (exception address) is ignored. Use "Location" options above to include/ exclude call stack's exception entry.

13. "**Call Stack: Full**" option will include all call stack items from current thread. Items will be included as: unit, class, function, line offset (e.g. same as with "**Location: Location**" option).

This will usually lead to duplicating exceptions, because call stack will be different on different machines/environments/OS.

Note: first item (exception address) is ignored. Use "Location" options above to include/ exclude call stack's exception entry.

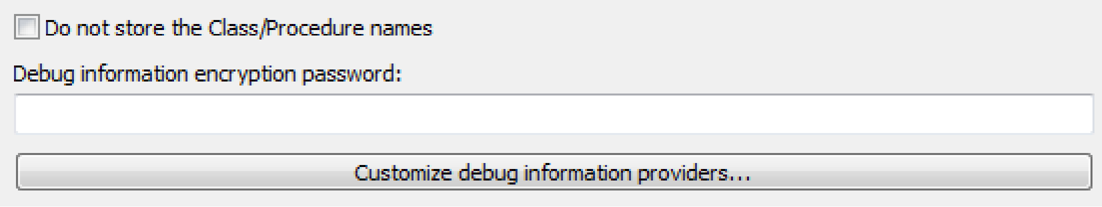
14. "**Allow EurekaLog to use user-defined "custom" low word**" option instructs EurekaLog to use CRC-32 (4 bytes) instead of CRC-16 (2 bytes). Enabling this option will give you less possible collisions, but disable custom part 421 of BugID.

Note: OnCustomBugID event will not be called if this option is turned on.

See also:
- BugID 421

10.3.2.3 Debug information page

This is "Debug information" page in EurekaLog project's options 225.



□ Do not store the Class/Procedure names

Debug information encryption password:

Debug information options

Options on "Debug information" page allow you to customize EurekaLog debug information options.

1. "**Do not store Class/Procedure names**" option disables storing names of classes, methods, functions and procedures inside executables. Unit names and line numbers are still saved.

Enable this option to minimize executable size and increase shareware protection.
Disable this option for best detalization.

2. "**Debug information encryption password**" option specifies password to encrypt all injected debug information. EurekaLog uses TEA cipher to encrypt all debug information.

If you set this option - you will not be able to view call stacks and assembly info in bug reports, it will be encrypted. To view encrypted report - you must use EurekaLog Viewer tool[617] and specify password.

Use this option if you don't want for end-users to view information about your executable.

Notes:

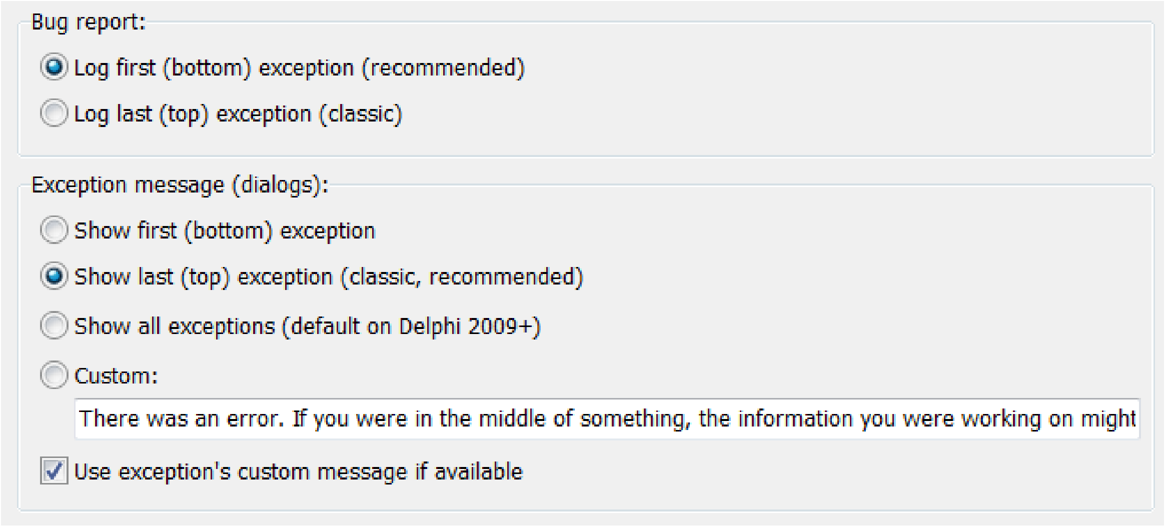
- Empty password does not mean that debug information will be injected in clear text. It still will be encrypted and packed (even with empty password). Empty password just allows you to view this information in error dialogs and bug reports in clear text. Non-empty password will prevent that.
- You can also supply a valid password to application at run-time to decrypt debug information on the go. When valid password is supplied - call stack in bug report and dialog will be displayed as clear text, even though debug information will still be stored encrypted. Currently there are three methods to specify decryption password at run-time:
  - o **EurekaLog automatically caches debug information passwords on your developer machine** (that is - on PC which was used to edit/save settings with password). That way call stacks will always be decrypted on your developer machine, because EurekaLog will retrieve decryption password from cache. Running your application on any other machine (e.g. production, clients, etc.) will show encrypted call stacks, because no decryption password is available.
  - o Use OnPasswordRequest event. You can either ask user, read registry, or return hard-coded password (however, do not hard-code passwords for production!).
  - o Use `--el_password=`*`your-password`* command-line option when launching your application.

See also:

- Configuring call stack[48]
- Debug information providers[409]
- EurekaLog for shareware developers[585]
- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.)[495]
- Using EurekaLog with non-Embarcadero DLLs (Microsoft Visual Studio, etc.)[496]

10.3.2.4 Nested exceptions page

This is "Nested exceptions" page in EurekaLog project's options[225].



Bug report:

● Log first (bottom) exception (recommended)

○ Log last (top) exception (classic)

Exception message (dialogs):

○ Show first (bottom) exception

● Show last (top) exception (classic, recommended)

○ Show all exceptions (default on Delphi 2009+)

○ Custom:

There was an error. If you were in the middle of something, the information you were working on might

☑ Use exception's custom message if available

Nested exceptions options

Chained exception is an exception which occurs during processing of another exception[40]. That "another" (original) exception is called "**nested exception**". For example:

```
try
  raise ERangeError.Create('Invalid item index'); // <- low-level error (a.k.a. ori
except
```

```
  raise EFileLoadError.CreateFmt('Error loading file %s', [FileName]); // <- high-l
end;
```

Note: for more information about nested/chained exeptions - see this article 573.

As you can see, low-level exception is the exception you're interested in. It indicates a reason for failure. This is what you typically want to be logged. Chained exception is triggered by original exception and provides more descriptive error message. So, you typically want to show it to user as error message. Thus, typically you want first exception to be logged, but last exception to be shown to end user.

Classic/default Delphi and C++ Builder behaviour is to work only with last exception always.

Delphi 2009+ only: starting with Delphi 2009 - there was new features introduced to exceptions in RTL. Support for chained exceptions was added. There are new properties BaseException and InnerException as well as special raising construct. In this model, you need to use RaiseOuterException or ThrowOuterException to preserve original exception when raising new exception. EurekaLog implements similar model with the same properties, except it doesn't require you to use special raising construct. Any exception raising automatically saves previous (original) exception in InnerException property. This feature available on all supported IDE versions.

Options on "Nested exceptions" page allow you to customize EurekaLog behaviour related to nested/chained exceptions.

Important note: this feature require EurekaLog to be able to track life-time of exception objects. Therefore, it's highly recommended that you enable the following options:

- "Enable extended memory manager" option 250
- "Use low-level hooks" option 259
- "Capture stack only for exceptions from current module" option 237

Otherwise it's recommended that you keep all options on this page into "Classic" position, or EurekaLog *may* show information about wrong exceptions.

1. "**Log first (bottom) exception**" (.NestedExceptionStack) option force EurekaLog to use original exceptions (nested root) for logging. This includes class, message, call stack, BugID and other exception-related properties to be stored in log file, shown in "detailed" dialogs and/or sent over network. This option doesn't affect any other visual appearance. If there is no nested exception - this option will have no effect, the behavior will be the same as if "**Log last (top) exception**" would be selected. This option is recommended option for typical application.

2. "**Log last (top) exception**" (.NestedExceptionStack) option force EurekaLog to use chained exceptions for logging. This includes class, message, call stack, BugID and other exception-related properties to be stored in log file, shown in "detailed" dialogs and/or sent over network. This option doesn't affect any other visual appearance. Use this option for backward-compatible behavior.

3. "**Show first (bottom) exception**" (.NestedExceptionMessage) option force EurekaLog to use original exceptions (nested root) for visual display. This includes error messages, dialogs and other visual interactions with user. This option doesn't affect logging. If there is no nested exception - this option will have no effect, the behavior will be the same as if "**Show last (top) exception**" would be selected.

4. "**Show last (top) exception**" (.NestedExceptionMessage) option force EurekaLog to use chained exceptions for visual display. This includes error messages, dialogs and other visual interactions with user. This option doesn't affect logging. This is recommended option for most application types.

5. "**Show all exceptions**" (.NestedExceptionMessage) option force EurekaLog to show messages from all currently active exceptions (each message on new line). This includes error messages, dialogs and other visual interactions with user. This option doesn't affect logging. This is default behavior of Delphi and C++ Builder applications starting with Delphi 2009, but it's generally not recommended (unless you want extra-detailed error display).

6. "**Custom**" (`.CustomExceptionMessage`) option allows you to override any exception message with custom string. This option is used for error messages, dialogs and other visual interactions with user. This option doesn't affect logging. Specifying a message will override any exception message with the specified string. Use this option to hide implementation details from end-user. Message overriding can also be done by using exception filters 185 or custom attributes 190.

7. "**Use exception's custom message if available**" (`.UseExceptionComments`) option allow you to override exception message with custom string individually for each exception. This option is used for error messages, dialogs and other visual interactions with user. This option doesn't affect logging. It can be combined with any of the previous options (items 3-6).

Each exception has associated "additional info" field. You can store any information in that field. It's like Tag field, except it has **String** type.

- If this option is checked and there is any value in that field - it will be used instead of actual exception message.
- If this option is unchecked or field is empty - Message property will be used.

You can use this feature to override original message with custom message, but still preserve original information. This override is used only for visual display. Log files are unaffected.

This feature allows you to use per-exception override (see also "Override Exception Message" option 344). If you want to override all messages - just use "**Custom**" option above.

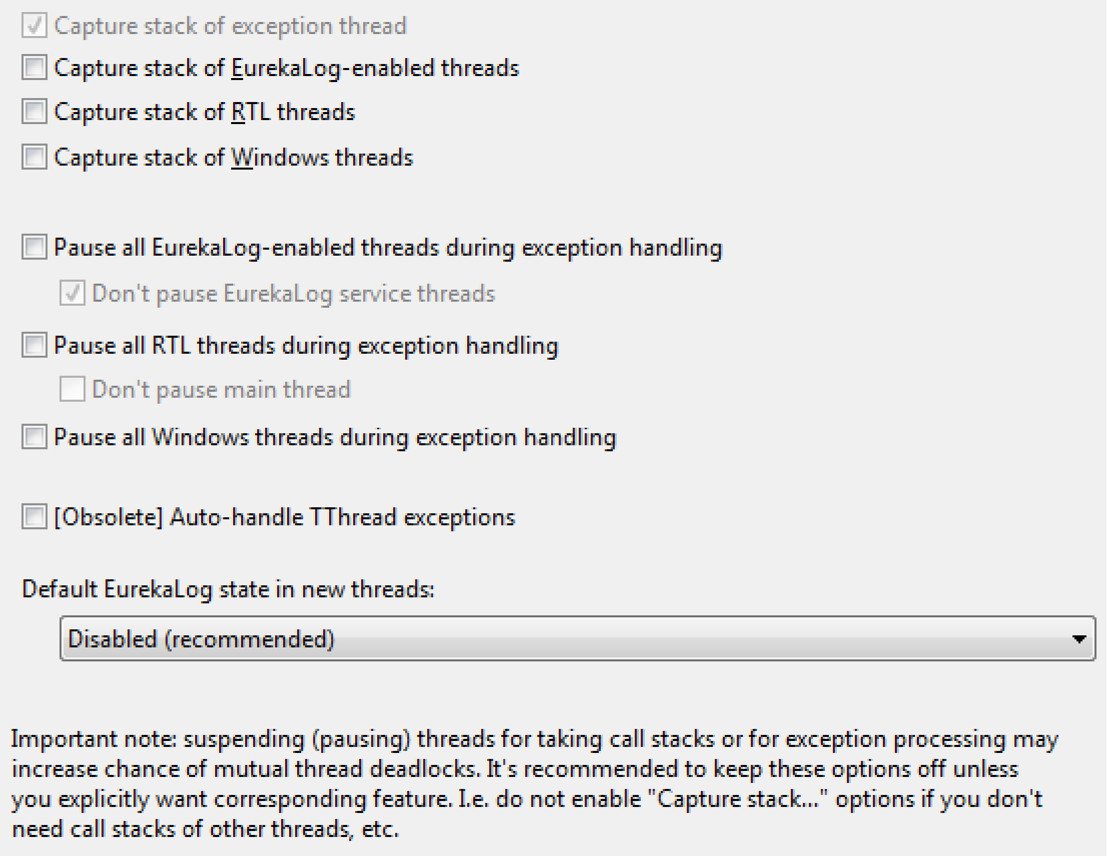
Note: some error dialogs like WER 389 and system logging 384 do not display error message to user (instead, they log exception details). Thus, such dialog are unaffected by the options above (items 3-7).

See also:

- Configuring call stack 48
- Chained exceptions 573

10.3.2.5 Multi-threading page

This is "Multi-threading" page in EurekaLog project's options 225.



☑ Capture stack of exception thread

□ Capture stack of EurekaLog-enabled threads

□ Capture stack of RTL threads

□ Capture stack of Windows threads

□ Pause all EurekaLog-enabled threads during exception handling

☑ Don't pause EurekaLog service threads

□ Pause all RTL threads during exception handling

□ Don't pause main thread

□ Pause all Windows threads during exception handling

□ [Obsolete] Auto-handle TThread exceptions

Default EurekaLog state in new threads:

Disabled (recommended)

Important note: suspending (pausing) threads for taking call stacks or for exception processing may increase chance of mutual thread deadlocks. It's recommended to keep these options off unless you explicitly want corresponding feature. I.e. do not enable "Capture stack..." options if you don't need call stacks of other threads, etc.

Multi-threading options

Options on "Multi-threading" page allow you to customize EurekaLog's multi-threading behavior.

1. "**Capture stack of EurekaLog-enabled threads**" (.csoShowELThreads) option includes call stacks of all EurekaLog-enabled threads in application - regardless of thread type. By default only exception thread is captured.

"EurekaLog-enabled thread" term refers to thread with enabled per-thread EurekaLog. You can enable EurekaLog in any thread by calling SetEurekaLogStateInThread function 570 or just simply create threads with TThreadEx 559 or BeginThreadEx 551. See Enabling EurekaLog for background threads 568 for more details.

Turn this option off for single-threaded application.
Turn this option on for multi-threaded application.

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will also require more time during exception processing.

2. "**Capture stack of RTL threads**" (.csoShowRTLThreads) option includes call stacks of all RTL threads in application. By default only exception thread is captured.

"RTL threads" means threads started with TThread or BeginThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread together with "Capture stack of EurekaLog-enabled threads" option instead.

Turn this option on to capture call stack of external RTL threads (that is threads started by 3rd party code without your control).

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will require more time during exception processing.

3. "**Capture stack of Windows threads**" (`.csoShowWindowsThreads`) option includes call stacks of all non-RTL threads in application. By default only exception thread is captured.

"Windows threads" means threads started with CreateThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread together with "Capture stack of EurekaLog-enabled threads" option instead. Alternatively, you may use "Capture stack of RTL threads" option instead.
Turn this option on to capture call stack of external Windows threads (that is threads started by 3rd party code without your control).
Never start your own thread with CreateThread function.

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will require more time during exception processing.

4. "**Pause all EurekaLog-enabled threads during exception handling**" (`.boPauseELThreads`) option will force EurekaLog to suspend all threads with enabled EurekaLog before processing (handling) exception, and resume these threads when processing (handling) will be completed.

Use this option if you don't want for other exception to occur in multi-threaded applications when you're displaying error dialog. Alternative to this option is to properly setup exception handling for threads (serialize) or to use restart options 259.

Note: in rare case suspending threads can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

5. "**Don't pause EurekaLog service threads**" (`.boDoNotPauseELServiceThread`) option excludes internal EurekaLog service threads (such as freeze detection thread, etc.) from list of threads for suspending.

6. "**Pause all RTL threads during exception handling**" (`.boPauseRTLThreads`) option will force EurekaLog to suspend all threads in your application created with TThread or BeginThread before processing (handling) exception, and resume these threads when processing (handling) will be completed.

Use this option if you don't want for other exception to occur in multi-threaded applications when you're displaying error dialog. Alternative to this option is to properly setup exception handling for threads (serialize) or to use restart options 259.

Notes:
- It is recommended to keep this option off and use "**Pause all EurekaLog-enabled threads during exception handling**" instead.
- In rare case suspending threads can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

7. "**Don't pause main thread**" (`.boDoNotPauseMainThread`) option excludes main thread from list of threads for suspending. Use this option if your event handlers are going to make synchronize calls into main thread.

Note: this option has no effect if "**Pause all RTL threads during exception handling**" and "**Pause all EurekaLog-enabled threads during exception handling**" options are not enabled.

8. "**Pause all Windows threads during exception handling**" (`.boPauseWindowsThreads`) option is equivalent to "**Pause all RTL threads during exception handling**" option, except this option will suspend/resume threads created with CreateThread function. Those threads are usually system service threads.

Use this option if you don't want for other exception to occur in multi-threaded applications when you're displaying error dialog. Alternative to this option is to properly setup exception handling for threads (serialize) or to use restart options 259.

Note: in rare case suspending threads can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

9. "**Auto-handle TThread exceptions**" option enables backward-compatible EurekaLog 6 behavior for threads. When you enable this option - EurekaLog will automatically handle exception in TThread. Default behavior is not to handle exception, but allow it to be saved in TThread.FatalException property, which can be analyzed/handled by caller thread.

Warning: enabling this option may result in multiple error dialogs at the same time (if several exception occur in multiple threads) and interfere with custom processing of TThread.FatalException property.

Notes:
- It's not recommended to use this option. You should implement proper error handling for threads instead. See Using EurekaLog in multi-threaded applications 547 for more details.
- EurekaLog has to be enabled for the background threads.

10. "**Default EurekaLog state in new threads**" option specifies default EurekaLog state for new background threads:
- If this option is set to "Enabled" - new threads will have EurekaLog enabled. It is not recommended value.
- If this option is set to "Disabled" - new threads will have EurekaLog disabled. It is recommended value.
- If this option is set to "Enabled for RTL threads, disabled for Windows threads" - threads created with BeginThread or TThread will have EurekaLog enabled, threads created with CreateThread will have EurekaLog disabled.

You can turn EurekaLog on/off on per-thread basis by using SetEurekaLogStateInThread function.

You can use this option to enable/disable EurekaLog for threads which you have no control of. Like system worker threads, thread pool, etc. You can always change per-thread EurekaLog state manually from your code.

It's highly recommended to keep this option into "Disabled" state and enable EurekaLog manually in your threads only - to avoid EurekaLog processing in 3rd party threads. See Using EurekaLog in multi-threaded applications 547 for more details.

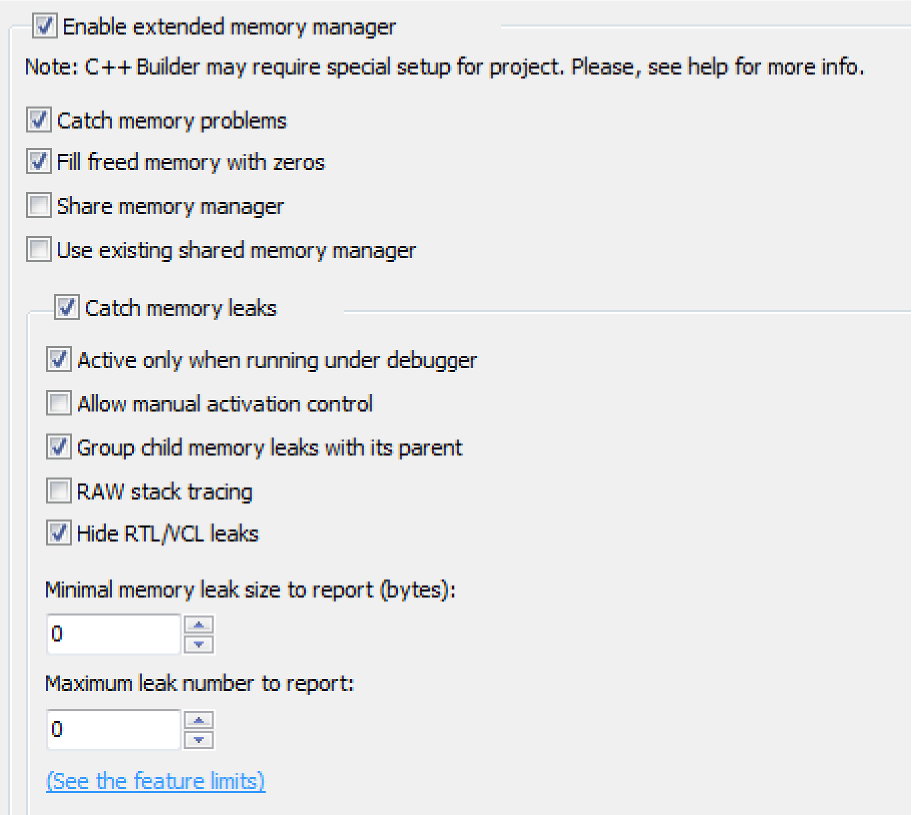
Important Note: Try to never use "Enabled" option as it will enable EurekaLog for all threads in your application - including internal system threads. Use "Enabled for RTL threads, disabled for Windows threads" instead of "Enabled" when possible.

See also:
- Using EurekaLog in multi-threaded applications 547

- Multi-threaded call stacks 85
- Configuring call stack 48

10.3.2.6 Memory problems page

This is "Memory problems" page in EurekaLog project's options 225.



☑ Enable extended memory manager

Note: C++ Builder may require special setup for project. Please, see help for more info.

☑ Catch memory problems

☑ Fill freed memory with zeros

□ Share memory manager

□ Use existing shared memory manager

☑ Catch memory leaks

☑ Active only when running under debugger

□ Allow manual activation control

☑ Group child memory leaks with its parent

□ RAW stack tracing

☑ Hide RTL/VCL leaks

Minimal memory leak size to report (bytes):

0

Maximum leak number to report:

0

(See the feature limits)

Memory debugging options

EurekaLog is capable to track many problems with heap memory (dynamically allocated memory).

1. "**Enable extended memory manager**" option enables EurekaLog's filter for memory manager. Installing debugging filter allow EurekaLog to perform additional checks. You must turn this option on, if you want to use debugging memory features of EurekaLog.

Notes:

- **(C++ Builder only):** C++ Builder requires special setup for this option to work. See using memory feature with C++ Builder 255.
- EurekaLog uses memory manager filter to track life-time of exception objects.
- This option is required for some other EurekaLog's features to work (like "Catch handled exceptions" and nested/chained exceptions).

Important note: it is recommended to always keep this option on, unless you need shared memory manager **and** need to communicate with module without EurekaLog's memory manager (or without EurekaLog at all) and without FastMM (EurekaLog is compatible with FastMM). **We highly do not recommend to turn this option off on RAD Studio 2007 and earlier.**

See also: using with shared memory manager 524.

You also need to enable at least 1 sub-option to get real work from this option. If you just enable "Enable extended memory manager" option, but don't select any sub-options - there will be no additional checks running.

2. "**Catch memory problems**" option enables checking for memory mis-use. Once wrong memory operation is detected - an exception will be raised. If you turn off this option - you code may perform invalid memory operation and continue running without noticing.

Memory mis-use includes the following:
- double-free (releasing same memory twice);
- cross-module operations (releasing memory allocated in other module - without using shared memory manager);
- heap buffer overflow (writing over memory block's borders);
- calling virtual methods of already deleted object;
- out of memory errors.

You can use EMemLeaks.MemLeaksErrorsToIgnore global variable to exclude certain checks (such as out of memory errors).

Enabling this feature has very little impact on the speed and memory consumption. It requires allocating few ten bytes more for each allocation.

Notes:
- it is recommended to always keep this option on;
- this option can work without leaks checking.

3. "**Fill freed memory with zeros**" option fills memory block with zeros on its release.

I.e. if this option is unchecked and you release a memory block - it will be left untouched (contains old data). So, if you try to access it again (bug, access after release) - this will be successful. If this option is checked and you release a memory block - this memory block will be erased (zeroed). Thus, any code which access memory block after release will get zeros instead of actual data - increasing chances to raise exception.

Enabling this feature has very little impact on the speed.

Notes:
- it is recommended to always keep this option on;
- this option can work without leaks checking;
- EurekaLog does not support checks for using memory blocks after free.

4. "**Share memory manager**" option enables installation of shared memory manager.

If you need to share memory manager or if you've already used shared manager before using EurekaLog - turn this option on. If there will be shared memory manager already present (installed by another module) - your module will use installed one. Your settings for memory manager will be ignored in this case.

This option allows sharing of the memory manager between .exe and DLLs which were also compiled with EurekaLog and this option enabled. This allows you to pass strings and dynamic arrays to DLL functions (and receive them from DLLs), provided that all modules are compiled with EurekaLog and this option enabled. Sharing will only work if the module that is supposed to share the memory manager was compiled with the "Share memory manager" or "Use existing shared memory manager" option set.

Notes:
- if .exe is single threaded and DLL is multi-threaded - then you must set the IsMultiThread variable in .exe to True, otherwise it will crash when a thread contention occurs.
- statically-linked DLL are initialized before .exe, so the main application may end up using memory manger from a statically loaded DLL and not the other way around.
- EurekaLog is capable of using FastMM's shared memory manager. You can safely mix modules with FastMM's shared memory manager and EurekaLog's shared memory manager. EurekaLog's memory features may be disabled in this case.
- if you're using shared memory manager - it's best to keep DLL and .exe compiled in the same Delphi or C++ Builder version. That's because format of internal data structures (like long string header) changes between compiler versions.

See also: using with shared memory manager 524.

5. "**Use existing shared memory manager**" option works the same as "Share memory manager" option, except it doesn't install shared memory manager, if there is no other shared memory manager set. I.e. your module will use installed shared memory manager, but will not install new one.

Note: use this option with caution, because statically-linked DLLs are loaded **before** .exe, so there may be unexpected initialization order. This leads to case, when some DLLs do not use installed shared memory manager. For this reason - consider using only "Share memory manager" option for common cases.

6. "**Catch memory leaks**" option enables memory leaks checking. Memory leak is a memory block which wasn't released. Usually it's a bug in your application.

If this option is unchecked - there will be no additional checks performed. If this option is checked and your application doesn't have any memory leak bugs - there will be no difference in application's behaviour. If this option is checked and your application has leaked the memory - there will be a usual EurekaLog's error message on application's shutdown. This error message will use dialog and send option as set in EurekaLog's options.

Note: it's recommended to always keep this and "Active only when running under debugger" options on.

Note: memory leak detection have some limitations 589.

7. "**Active only when running under debugger**" option alters activation behaviour of "Catch memory leaks" option.

If this option is checked - memory leak detection will only be activated when running application under debugger (like IDE). This can be useful if you want to debug your applications on developer's machine, but do not want to annoy end-users with useless error reports (because in most cases memory leaks don't bother end-users).

Note: it's recommended to always keep this option on, unless you're creating a server or service application.

8. "**Allow manual activation control**" option alters activation behaviour of "Catch memory leaks" option. If this option is checked - memory leak detection can be manually turned on or off via command-line switches.

Use "`/EL_EnableMemoryLeaks`" to enable memory leak detection.
Use "`/EL_DisableMemoryLeaks`" to disable memory leak detection.
Use "`/EL_DisableMemoryFilter`" to disable extended memory manager (see option #1 above).

If this option is unchecked - these command-line switches are ignored. If this option is checked - these command-line switches will turn on or off memory leaks detection. Command-line switches will override any other check for memory leak detection activation.

This option can be used with "Active only when running under debugger" option.

This option is useful, when you want to temporary enable memory leaks detection - for example, you can activate "Active only when running under debugger" option and deploy your application to end customers without bothering them with leaks error report. However, if some customer will report problems with memory usage - you can always advise to use command-line switch to enable memory leaks detection.

Note: it's not possible to enable extended memory manager via command-line switches.

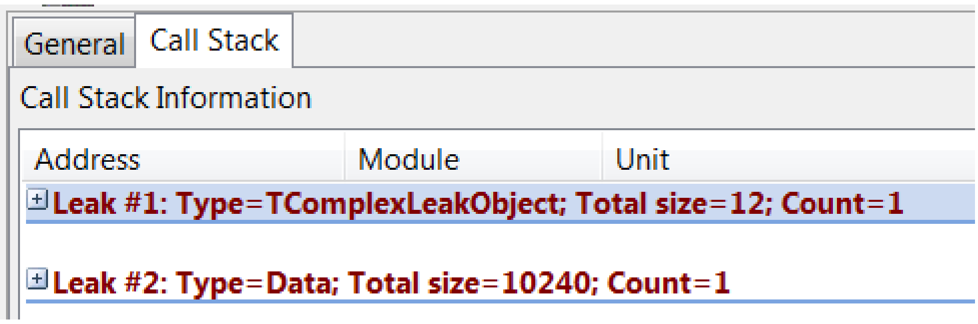
9. "**Group child memory leaks with its parent**" option hides child leaks from reports.

Leak of object is parent leak. Leak of object's field is child leak. For example:

```
type
```

```
  TComplexLeakObject = class
  private
    FData: Pointer;
  public
    constructor Create;
    destructor Destroy; override;
  end;

{ TComplexLeakObject }

constructor TComplexLeakObject.Create;
begin
  inherited;

  FData := AllocMem(10240);
end;

destructor TComplexLeakObject.Destroy;
begin
  if Assigned(FData) then
  begin
    FreeMem(FData);
    FData := nil;
  end;

  inherited;
end;

var
  LeakedObject: TComplexLeakObject;
begin
  LeakedObject := TComplexLeakObject.Create;
end;
```

This code will create 2 leaks: 1 for `LeakedObject` and 1 for `LeakedObject.FData`. If "Group child memory leaks with its parent" option is unchecked - you'll get report about 2 different leaks:

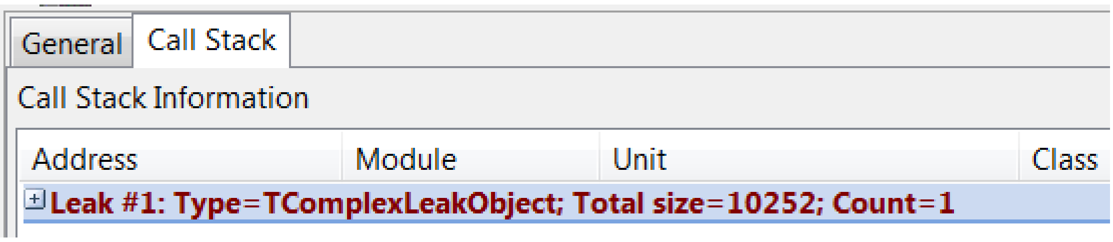


Leak #1: Type=TComplexLeakObject; Total size=12; Count=1

Leak #2: Type=Data; Total size=10240; Count=1

"Group child memory leaks with its parent option" is unchecked

If this option is checked - you'll get report only about 1 leak (`LeakedObject`). `LeakedObject.FData` will be hidden and size of `LeakedObject.FData`'s leak will be added to `LeakedObject`'s leak:



Call Stack Information

Address	Module	Unit	Class
⊞ Leak #1: Type=TComplexLeakObject; Total size=10252; Count=1			

"Group child memory leaks with its parent option" is checked

- This option is heuristic.
- Turn this option on to minimize large reports.
- Turn this option off to get detailed reports.

Note: this option is able to group leaks which are only created within constructor of parent object. If you assign object's fields outside of the constructor - these leaks will not be grouped.

10. "**RAW stack tracing**" option switches EurekaLog to use RAW tracing method for building call stacks for memory issues. By default EurekaLog uses frame-based tracing method.

RAW stack tracing can show more complete call stacks, but it's *significantly* slower. See also: RAW method and frame-based method 578.

Turn on for best detalization.
Turn off for best performance.

Note: it's recommended keep this option unchecked, until you face problem which you can't solve without more detailed call stack.

Warning: do not ship your product to clients with "RAW stack tracing" option enabled. Performance of your application will be significantly lower.

11. "**Hide RTL/VCL leaks**" option hides known RTL/VCL leaks. There are some leaks which are "by design" or are bugs in old Delphi/C++ Builder versions. You can't fix them, but you can ignore them.

Turn on to hide expected leaks or leaks from RTL/VCL bugs.
Turn off for better detalization.

Note: if you found another "known bug" of Delphi/C++ Builder - please, send us a description and demo, so we'll be able to exclude this leak from reporting and improve our "Hide RTL/VCL leaks" option.

12. "**Minimal memory leak size to report**" option allows you to ignore small leaks.

If you set this value to 0 - EurekaLog will report all leaks. If you set it to some value > 0 - EurekaLog will only report leaks, if total leaked memory's size is greater than (or equal to) the specified value.

Set to 0 to increase detalization.
Increase value to decrease report's noise.

13. "**Maximum leak number to report**" option allow you to limit count of reported memory leaks. This is useful option, if situation is so bad so your application generates huge amount of leaks.

If you set this value to 0 - EurekaLog will report all leaks (use with caution). If you set it to some value > 0 - EurekaLog will only report this number of leaks, ignore all other leaks. I.e. EurekaLog will report only first <N> leaks, where N is entered value.

Use this option to limit size of memory leak reports, to avoid creation of huge bug reports (and avoid temporary hung at shutdown, when EurekaLog will busy creating report).

Increase value to increase detalization level.

Decrease value to reduce noise and decrease report's size.

See also:
- Enabling memory features for C++ Builder 255
- Configuring project for leaks detection 508
- Other additional features 237
- Memory leaks limitations 589
- Solving memory leaks 166
- Solving memory problems 171
- Using EurekaLog and 3rd party shared memory manager 524

10.3.2.6.1 Enabling memory/resource leaks features for C++ Builder

In order to use EurekaLog's memory features (like heap corruption checks or memory leaks detection) or resource leak detection feature in your C++ Builder application - you must perform special setup of your application.

Please, follow these steps:
1. Enable memory features 250 or resource features 255 in EurekaLog's options for your project 225.
2. (for memory leaks only) Go to "Project"/"Options"/"C++ Linker" and set "**Dynamic RTL**" option to **False**.
3. (optional, recommended) Go to "Project"/"Options"/"Packages" and set "**Build with run-time packages**" option to **False**.
4. Copy **EMemLeaksBCB.cpp** file to your project folder, if it doesn't exist yet (this file is originally located in \Lib\Common subfolder of your EurekaLog installation).
5. Add **EMemLeaksBCB.cpp** file from your project's folder to the project.
6. Go to "Project"/"Options"/"Build order" and move **EMemLeaksBCB.cpp** file to be the **first** unit (on top).
7. Open **project's source** ("Project"/"View source") and **add** `USEOBJ("EMemLeaksBCB.cpp")` before any `USEFORM` directives, for example:

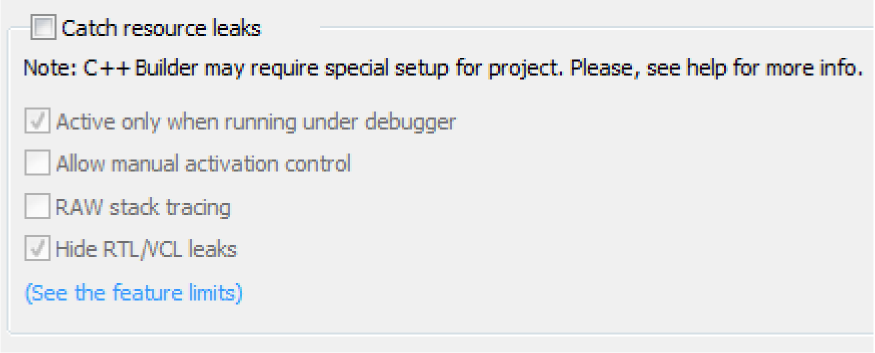
```
//---------------------------------------------------------------------------

#include <vcl.h>
USEOBJ("EMemLeaksBCB.cpp");   // <- added
#pragma hdrstop
#include <tchar.h>
//---------------------------------------------------------------------------
USEFORM("Unit9.cpp", Form9);
//---------------------------------------------------------------------------
...
```

8. Make a full build ("Project"/"Build"); not just compile or make.

See also:
- Memory features settings 250
- Resource leaks settings 255
- Memory leaks detection limitations 589
- Resource leaks detection limitations 589

10.3.2.7 Resource leaks page

This is "Resource leaks" page in EurekaLog project's options 225.



□ Catch resource leaks

Note: C++ Builder may require special setup for project. Please, see help for more info.

☑ Active only when running under debugger

□ Allow manual activation control

□ RAW stack tracing

☑ Hide RTL/VCL leaks

(See the feature limits)

Resource leaks detection options

EurekaLog is capable to track resource leaks - dynamically allocated resources other than memory 250.

1. "**Catch resource leaks**" option enables resource leaks checking. Resource leak is a handle of some object which wasn't released. Usually it's a bug in your application.

If this option is unchecked - there will be no additional checks performed. If this option is checked and your application doesn't have any resource leak bugs - there will be no difference in application's behaviour. If this option is checked and your application has leaked the resource - there will be a usual EurekaLog's error message on application's shutdown. This error message will use dialog and send option as set in EurekaLog's options.

Note: resource leak detection have some limitations 589.

Note: resource leaks detection works only with supported functions 590.

2. "**Active only when running under debugger**" option alters activation behaviour of "Catch resource leaks" option.

If this option is checked - resource leak detection will only be activated when running application under debugger (like IDE). This can be useful if you want to debug your applications on developer's machine, but do not want to annoy end-users with useless error reports (because in most cases resource leaks don't bother end-users).

Note: it's recommended to always keep this option on, unless you're creating a server or service application.

3. "**Allow manual activation control**" option alters activation behaviour of "Catch resource leaks" option. If this option is checked - resource leak detection can be manually turned on or off via command-line switches.

Use "`/EL_EnableResourceLeaks`" to enable resource leak detection.
Use "`/EL_DisableResourceLeaks`" to disable resource leak detection.

If this option is unchecked - these command-line switches are ignored. If this option is checked - these command-line switches will turn on or off resource leaks detection. Command-line switches will override any other check for resource leak detection activation.

This option can be used with "Active only when running under debugger" option.

This option is useful, when you want to temporary enable resource leaks detection - for example, you can activate "Active only when running under debugger" option and deploy your application to end customers without bothering them with leaks error report. However, if some customer will report problems with memory/resource usage - you can always advise to use command-line switch to enable resource leaks detection.

Note: it's recommended to always keep this option on.

4. "**RAW stack tracing**" option switches EurekaLog to use RAW tracing method for building

call stacks for resource leaks. By default EurekaLog uses frame-based tracing method.

RAW stack tracing can show more complete call stacks, but it's *significantly* slower. See also: RAW method and frame-based method 578.

Turn on for best detalization.
Turn off for best performance.

Note: it's recommended keep this option unchecked, until you face problem which you can't solve without more detailed call stack.

Warning: do not ship your product to clients with "RAW stack tracing" option enabled. Performance of your application will be significantly lower.

5. "**Hide RTL/VCL leaks**" option hides known RTL/VCL leaks. There are some leaks which are "by design" or are bugs in old Delphi/C++ Builder versions. You can't fix them, but you can ignore them.

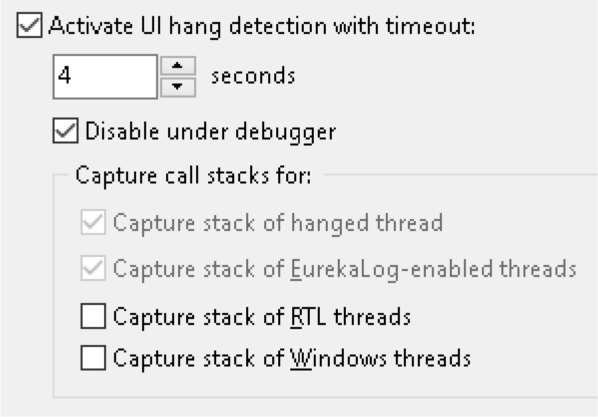
Turn on to hide expected leaks or leaks from RTL/VCL bugs.
Turn off for better detalization.

Note: if you found another "known bug" of Delphi/C++ Builder - please, send us a description and demo, so we'll be able to exclude this leak from reporting and improve our "Hide RTL/VCL leaks" option.

See also:
- Enabling resource features for C++ Builder 255
- Configuring project for leaks detection 508
- Other additional features 237
- Resource leaks limitations 589

10.3.2.8 Hang detection page

This is "Hang detection" page in EurekaLog project's options 225.



☑ Activate UI hang detection with timeout:
4 seconds
☑ Disable under debugger
Capture call stacks for:
☑ Capture stack of hanged thread
☑ Capture stack of EurekaLog-enabled threads
☐ Capture stack of RTL threads
☐ Capture stack of Windows threads

Hang detection options

This page allows you to configure hang and deadlock detection feature in EurekaLog.

1. "**Activate UI hang detection with timeout**" option enables EurekaLog to monitor running application for hangs 174. A timeout (in seconds) indicate how long UI must be unresponsive to be detected as "hang".

High values will result in longer hang detection.
Low values will result in false-positive triggering.

Recommended value is 4 seconds.

Notes:

- System default timeout for hang detection is 5 seconds. Thus system hang detection dialog may be triggered before EurekaLog's hang detection, if you would set timeout value to large values (larger than 5 seconds).
- EurekaLog is able to detect hang of only the main thread on pre-Vista systems. EurekaLog uses SendMessageTimout function to detect whenever main thread is not able to receive messages (i.e. it hangs).
- EurekaLog is able to detect deadlock between any EurekaLog-enabled threads on Vista+ systems. EurekaLog uses Wait Chain Traversal (WCT) to detect whenever there is a deadlock between two or more EurekaLog-enabled threads.
- All EurekaLog-enabled threads are included in hang bug report - regardless of "Capture stack of EurekaLog-enabled threads" option 246.

2. **"Disable under debugger"** (`.FreezeDisableUnderDebugger`) option will disable hang detection feature if you run your application under debugger. The hang detection will be enabled when application run outside of IDE/debugger.

Note: this option has no effect if "Activate UI hang detection with timeout" option is not checked.

Important note: it is recommended to enable this option when debugging your application. Time spend in the debugger will be accounted for non-responding time of your application. Thus, a hang report may be triggered while you are debugging your application. Use this option to avoid such false-positive reports.

3. "**Capture stack of RTL threads**" (`.csoShowRTLThreads`) option includes call stacks of all RTL threads in application. By default only hanged thread and EurekaLog-enabled threads are captured.

"RTL threads" means thread started with TThread or BeginThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread.
Turn this option on to capture call stack of external RTL threads (that is threads started by 3rd party code without your control).

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

Taking call stack of additional threads will require more time during exception processing.

4. "**Capture stack of Windows threads**" (`.csoShowWindowsThreads`) option includes call stacks of all non-RTL threads in application. By default only hanged thread and EurekaLog-enabled threads are captured.

"Windows threads" means thread started with CreateThread.

It is recommended to keep this option off and use TThreadEx 559 and BeginThreadEx 551 or SetEurekaLogStateInThread. Alternatively, you may use "Capture stack of RTL threads" instead.
Turn this option on to capture call stack of external Windows threads (that is threads started by 3rd party code without your control).

Note: capturing call stack of an external thread requires thread's suspending. In rare case this can cause deadlock issues (for example: thread may be suspended when it is running memory allocation function; thus, any further memory alloc/release operation will block application forever). Do not enable this option until really needed. See Using EurekaLog in multi-threaded applications 547 for more details.

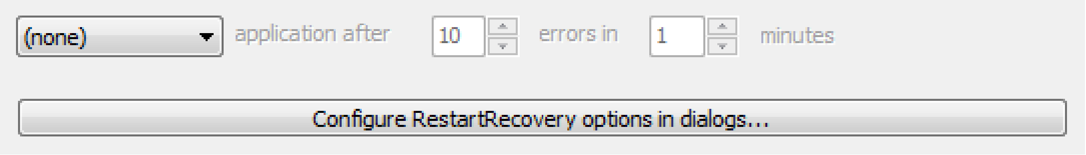
Taking call stack of additional threads will require more time during exception processing.

See also:

- Hangs and deadlocks 174

10.3.2.9 Restart&Recovery page

This is "Restart&Recovery" page in EurekaLog project's options 225.



Restart and recovery options

Options on "Restart&Recovery" page allow you to customize EurekaLog behavior for restarting application on errors.

1. "**None/Restart/Terminate**" (`.AutoCrashOperation`, `.AutoCrashNumber`, `.AutoCrashMinutes`) option specifies action to trigger when more than specified number of errors occurs in less than specified amount of time interval. **None** option will disable this feature. **Restart** option will automatically restart application. **Terminate** option will automatically close application.

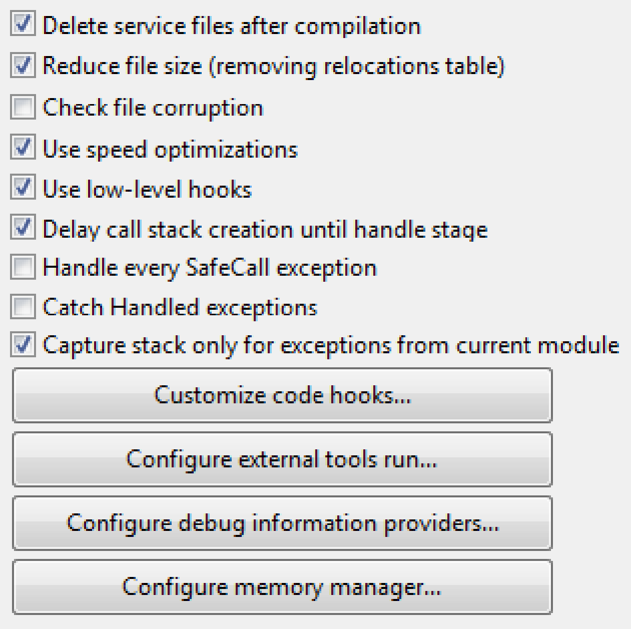
Use this option to automatically restart/terminate application in case of "exception spam" - i.e. when large numbers of exception occurs in a short amount of time.

Notes:

- This option is independent from error dialogs. If you want restart options to be controlled by end-user - then setup appropriate options for dialogs, do not enable this feature. It's not recommended to enable both options (restart in dialogs and this feature) to avoid confusion for end-users.
- When number of errors is set to 1 - application will be restarted/terminated after first exception regardless of timeout.
- Only unhandled exceptions (i.e. processed by EurekaLog) are counted. If exception is not handled by EurekaLog - it will not be counted. Thus, it's perfectly possible to build application with heavy exception usage.
- This is global counter. Be extra careful to use it in multi-threaded applications, as exceptions from multiple threads will be counted within the same timeout interval. For example, if you set to terminate application after 10 errors within 1 minute, and you have 10 threads, and each thread will raise 1 exception - then your application will be terminated.
- This setting specify "time window" to monitor exceptions. It is designed to close application if it is "rushed" with exceptions. This is very similar to "Show restart checkbox after N error" setting in dialogs. For example, if there is 10 exceptions total, one exception each second - then application will be terminated after 10 seconds, even though this setting says "Terminate after 10 errors in 1 minute" - because this is what just happened: you get your 10 exception within 1 minute time window.
- EurekaLog may terminate your application regardless of these options when it encounters a fatal problem. For example, out of memory error or memory corruption will generate bug report and terminate your application - because it is not possible to continue normal execution after memory problems.

10.3.2.10 External tools

This is "External tools" page in EurekaLog project's options 225.



☑ Delete service files after compilation
☑ Reduce file size (removing relocations table)
□ Check file corruption
☑ Use speed optimizations
☑ Use low-level hooks
☑ Delay call stack creation until handle stage
□ Handle every SafeCall exception
□ Catch Handled exceptions
☑ Capture stack only for exceptions from current module

Customize code hooks...
Configure external tools run...
Configure debug information providers...
Configure memory manager...

External tools options

Options on "External tools" page allow you to customize EurekaLog behavior for integration with other 3rd party software. This page contains options from other pages.

1. "**Delete service files after compilation**" option allows you to clean up your project from unnecessary files. EurekaLog asks linker to generate .map/.tds/.drc files to read debug information from them. Once debug information is injected - these files are no longer needed. This option can be used to delete them automatically. See EurekaLog's basics 38 for more information. You can turn this option off to keep files untouched (if you want them for other 3rd party software).

Turn this option off for external debuggers, profilers and debug information convertor tools (such as map2dbg).

2. "**Reduce file size**" option removes relocation table from file. This reduces file's size for about 10% (however, enabling EurekaLog also increases file's size).

It's recommended to always keep this option on.

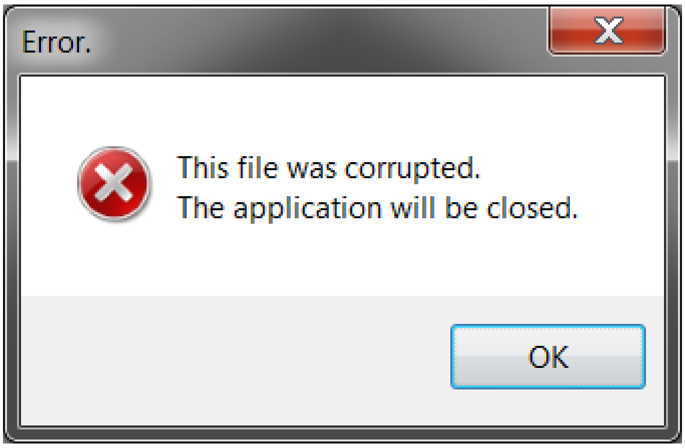
Note: this option have no effect for DLL and packages.

Technical explanation
When you compile a DLL (or a package which is a Delphi-specific DLL in disguise), the linker includes what is known as a relocation table. This table includes information about what addresses must be fixed up by the OS loader in the (likely) event that the DLL must be loaded at a different address than its intended-at-compile/link-time base address. You see, all DLLs come with a base address that is the "ideal" loading address of that module. The OS will try to load the DLL at this address to avoid the overhead of runtime rebasing (patching the in-memory pages of the DLL forces it to be paged in from disk and prevents cross-process sharing of the DLL pages). That's why you should set the Image base option in the Linker page of the project options of DLL and package projects. The default Image base that Delphi uses is $00400000 for both applications, DLLs and packages - and thus by default all DLLs and packages will need to be rebased - as that address is reserved for the process' EXE file.

The implication is that an EXE file will always be loaded at the fixed virtual address $00400000 and that it will never need to be rebased. Alas, it doesn't really need its relocation table and we can safely remove it, shrinking the size of the .EXE without affecting its behavior or performance.

3. "**Check file corruption**" option adds check for file corruption in your project. If you enable

this option, EurekaLog will calculate a CRC checksum of the compiled file and store it inside file. EurekaLog will also read this checksum from file on its startup (launch). If your executable was modified, EurekaLog will display an appropriate message and shutdown your application immediately:



Error.

This file was corrupted.
The application will be closed.

EurekaLog detected changes in executable file

You can use this option to ensure that your code wasn't modified.

Notes:

- `CheckSum` field in the `IMAGE_OPTIONAL_HEADER` structure is used to store CRC value inside executable file.
- This option checks file on disk, not running process image.
- Enabling this option will slow down loading and startup times on your executable. The bigger your executable file will be - the larger will be startup time: because the entire file must be read at startup.

Turn off for digitally signed files, packers or protectors.

4. "**Use speed optimizations**" option enables caching of kernel information to improve performance of stack tracing.

Turn off for packers and protectors.

Note: this option will be automatically disabled when "Detalization level" option 237 is set to "Show any (including RAW addresses)" or "Show any item belong to executable module (unknown locations within DLL)" value.

It is recommended to keep this option unchecked unless you suffer from slow work of your application.

5. "**Use low-level hooks**" option allows or forbids using of low-level hooks.

Using low-level hooks allows you to capture low-level information such as CPU state. Low-level hooks are also required for additional WER functionality. However, a documented way of installing low-level hook is available only in Windows XP and later. For older OS - undocumented hack will be used. If this option is unchecked - EurekaLog will use RTL functionality and will not install low-level hooks.

Notes:

- Low-level hooks will always be used on Delphi 2007 and below, since RTL support for handling exception was only introduced in Delphi 2009.
- Using of low-level hooks may introduce compatibility issues with 3rd party protection software with anti-crack detection.
- EurekaLog uses different implementation on Windows 2000 and Windows XP and above:
  - Windows 2000: use SEH - inject hook into `KiUserExceptionDispatcher` (undocumented hack).
  - Windows XP+: use VEH - add handler via documented API.

- This option controls only collecting information stage ("raise"). This option has no effect on other places. For example, hooks for handling exception are controlled by these options 352.

Turn off for packers and protectors.

Important note: turning off low-level hooks means that EurekaLog will not install additional hooks for API functions. This means that EurekaLog will not intercept important system calls. For example, EurekaLog will not hook ExitThread function, which means EurekaLog will not know when a thread exits. This will lead to thread information stored forever - until application terminates. You can call internal _NotifyThreadGone or _CleanupFinishedThreads functions to notify EurekaLog about thread's termination. Such manual notifications can be avoided by using EurekaLog's wrappers (TThreadEx 559, for example).

6. "**Delay call stack creation until handle stage**" option postpones analyzing exception for later stages of processing.

Please, see this article 583 for detailed explanation of delayed (deferred) call stacks.

Enable this option for better performance.
Disable this option for better detalization and compatibility.

Note: enabling some advanced features of EurekaLog (such as handling safecall exceptions, using exception filters with "Exception Kind" <> "All", etc.) may require creating call stack earlier than usual (for example: to detect if exception is raised within safecall wrapper), so this option will have no effect.

7. "**Handle every SafeCall exception**" option is used to catch safecall-exceptions with EurekaLog. This option is useful in COM servers, COM applications and other interface-related code.

When this option is off - safecall exceptions will be handled by default processing which usually means losing information about error location.
When this option is on - safecall exceptions will be handled by EurekaLog and then by default processing.

Usually it's a good idea to disable error dialogs and visual feedback for safecall exceptions since these exception will be handled by calling code (which will display error message).

Notes:
- Each safecall exception is considered to be handled exception. Keep that in mind when you setup exception filters or write event handlers.
- This option has no effect if "**Catch handled exceptions**" option is enabled (see below).
- This option requires extended memory manager 250 enabled.
- It's a good idea to include fix for QC report #81725 352 when you use "**Handle every SafeCall exception**" option.
- Internally, "**Handle every SafeCall exception**" option installs hook for ComObj.HandleSafeCallException routine (when low-level hooks are allowed) or scans exception's call stack for _HandleAutoException routine (when low-level hooks are not installed). The later can cause building call stack for all exceptions even with "deferred call stacks" option set.

Alternative for this option is to invoke EurekaLog manually from your SafeCallException handler.

See also:
- Using EurekaLog in COM applications 488

8. "**Catch handled exceptions**" option will enable EurekaLog for all exceptions. By default EurekaLog processes only exceptions which are unhandled (see handled/unhandled terms definitions 40).

It's not recommended to use this option. That's because "handled" for exception means that this exception is expected and it was handled by code. Therefore, it's better to setup proper exception handling in your code. This option is used primary as last resort measure to work with "bad" code (the code which hides unhandled exceptions).

Be sure to setup proper exception filtering when you enable this option. Often it's a good idea to disable error dialogs and visual feedback for handled exceptions.

Notes:
- You should use "**Handle every SafeCall exception**" option for safecall-exceptions instead of "**Catch handled exceptions**" option.
- This option requires extended memory manager 250 enabled.

9. "**Capture stack only for exceptions from current module**" (`.csoCaptureOnlyModuleExceptions`) option allows you to speed up execution by ignoring all exceptions outside of your executable module.

Since normal practice for exceptions is to handle them within the same module 457 - exceptions usually do not leave module (i.e. they are not shared between modules). This means that you're usually interested only in exceptions from the same module. This option allows you to ignore any other exception.

Note: this option is extremely useful in applications with plug-ins (including COM modules).

It's recommended to keep this option checked when possible. Disable this option for packaged applications or other application types which includes sharing exceptions between modules.

10. "**Customize code hooks**" option allows you to select code hooks 352 specific for application type 363. Disable hooks for packers and protectors.

11. "**Configure external tools run**" option allows you to specify commands to invoke during building process 351. You can place a call to external tool (such as packer, protector or debug information converter).

Delphi 2009+: order of actions during project's compilation is as follows:
1. EurekaLog pre-build event
2. IDE pre-build event
3. Project compile and link
4. IDE post-build event
5. EurekaLog link (post-processing)
6. EurekaLog post-build event

Note: you must post-process executable with EurekaLog **first**, and only after this - you can post-process executable with packer/protector/digital signature tool. On the other hand, debug information converter may be run before or after EurekaLog's post-processing - it doesn't matter (as long as debug information is present and untouched).

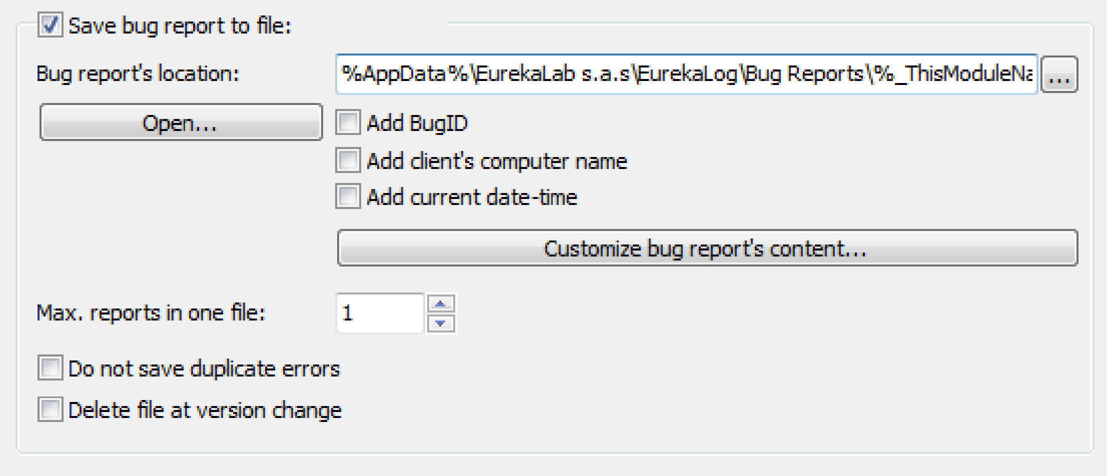
12. "**Customize debug information providers**" option allows you to customize debug information reader classes ("providers") 355. You may want to enable additional providers to be able read debug information supplied by external tools.

13. "**Customize memory manager**" option allows you to customize debugging memory manager 250. It's highly recommended to keep "**Enable extended memory manager**" option turned on (you can disable other memory checking options if you want to). Installing filter on memory manager will allow EurekaLog to track life-time of exceptions objects without need to install code hooks.

See also:
- Using EurekaLog with external software 514
- Using EurekaLog in COM applications 488
- Code hooks 352
- Build events 351
- Debug information providers 355

10.3.3 Bug report page

This is "Bug report" page in EurekaLog project's options 225.



☑ Save bug report to file:

Bug report's location: %AppData%\EurekaLab s.a.s\EurekaLog\Bug Reports\%_ThisModuleNa

□ Add BugID

□ Add client's computer name

□ Add current date-time

Max. reports in one file: 1

□ Do not save duplicate errors

□ Delete file at version change

Bug report options

Bug report page contains local exception logging options. You can enable/disable saving bug reports to file here. See this article 72 for more information about bug reports.

1. "**Save bug report to file**" (`.SaveLogFile`) option enables saving full exception information report into a file. This file is plain-text report file, which can be viewed in any text editor or special viewer application: EurekaLog Viewer. Bug report contains information specified on Bug report content 266 page. Single file can contains multiple problem reports.

2. "**Bug report's location**" (`.OutputPath`) option specifies path on file system, where you want to save bug report. Default is a project's subfolder in "`%APPDATA%\Neos Eureka S.r.l\Bug reports\`" (see also 217). Empty string means default path (i.e. the same folder - "`%APPDATA%\Neos Eureka S.r.l\Bug reports\`"). You can use environment variables 413 to specify dynamic-changed paths.

This option can specify folder only (ends with path separator - '/'). In this case: default file name ("*your-project*.el" - for example: "Project1.el") will be used. Alternatively, you can specify both path + file name. In this case: specified custom file name will be used. Note that you also can alter file name by using the following 3 options (see below).

If you want to save bug report to the same folder as executable module - use "`.\`" folder. You can also use any other relative path, like this for example: "`.\Reports`". Do not save bug report to .exe's folder, if you install your application to Program Files. Use exe's folder only for mobile applications (for example: runnable from flash sticks).

Notes:
1. You can open default folder with bug reports by opening **Start** / **Programs** / **EurekaLog** / **Bug reports** menu item (for current user account only).
2. If your selected folder will be write-protected at run-time, EurekaLog will revert it to default. If path doesn't exist - it will be created.

Important Note: `%APPDATA%` folder is specific to user account. Each user account has its own `%APPDATA%` folder. EurekaLog will use `%APPDATA%` folder of user running your executable. For example, if you are writing a Win32 Service application - such application is run by "`Local System`" account by default. "`Local System`" account has its own `%APPDATA%` folder, for example: "`C:\WINDOWS\system32\config\systemprofile\AppData`" or "`C:\WINDOWS\SysWOW64\config\systemprofile\AppData`" (depending on bitness of your application and operating system).

3. "**Add BugID**" (`.loAddBugIDInLogFileName`) option alters default file name by appending a Bug ID 421 value to it. Bug ID is a hash value of type and location of the problem. Exceptions

with the same Bug ID is considered to be the same.

Use this option to generate a more customized/unique bug report's file names.

Example: "`Project1_A5810000.el`".

Warning: checking this option will result in multiple log files for your project (one file for each exception kind). It's recommended to keep this option checked off (unchecked).

Note: there is a similar option 304 which is applied to name of send bug report only. You may consider using it instead of this option.

4. "**Add client's computer name**" (`.loAddComputernameInLogFileName`) option alters default file name by appending a name of client's machine value to it. All non-allowed file name characters will be replaced with safe replace character ('_').

Use this option to generate a more customized/unique bug report's file names.

Example: "`Project1_Alex_Notebook.el`".

Note: there is a similar option 304 which is applied to name of send bug report only. You may consider using it instead of this option.

5. "**Add current date-time**" (`.loAddDateInLogFileName`) option alters default file name by appending a current date-time value to it ('yyyymmddhhnnss').

Use this option to generate a more customized/unique bug report's file names.

Example: "`Project1_20110609005134.el`".

Warning: checking this option will result in multiple log files for your project (one file for each exception occurrence). It's recommended to keep this option checked off (unchecked).

Note: there is a similar option 304 which is applied to name of send bug report only. You may consider using it instead of this option.

6. "**Max. reports in one file**" (`.ErrorsNumberToSave`) option allows you to limit capacity of a single file. It specifies how many reports can be hold in 1 reports. Once limit is exceeded, oldest report in file will be deleted and new report will be saved as last report. Value of 0 means unlimited bug report file. Typical values can be 0, 1, 32, 256, 9999, etc.

Setting this option to 1 is almost equivalent of (non-existed) "delete log file before exception" option.

7. If you enable the "**Do not save duplicate errors**" (`.loNoDuplicateErrors`) option - then file will contain only unique bug reports, count field (2.8) 266 will contain number of occurrences for each exception (if fields is present, otherwise duplicate count information is omitted and lost). When this option is unchecked - all reports will be saved, count field (2.8) will always be 1 for all reports.

Notes:
- Report uniqueness is established via Bug ID 421 property. Bug ID is a hash value of type and location of the problem. Exceptions with the same Bug ID is considered to be the same.
- Usually you should set a limit for maximum reports in one file greater than 1 to get useful results from this option. If you set limit to 1 - this option will have almost zero effect.
- This option doesn't require count field (2.8) to be included into bug report. However, usually you use these two options at the same time.
- This option affects only local bug report file. It does not affect sending 302.

8. "**Delete file at version change**" (`.loDeleteLogAtVersionChange`) option deletes bug report file, when application's version change. This is useful when you store multiple reports in a single file and don't want a "noise" from old version of your application.

This option has no effect, if:

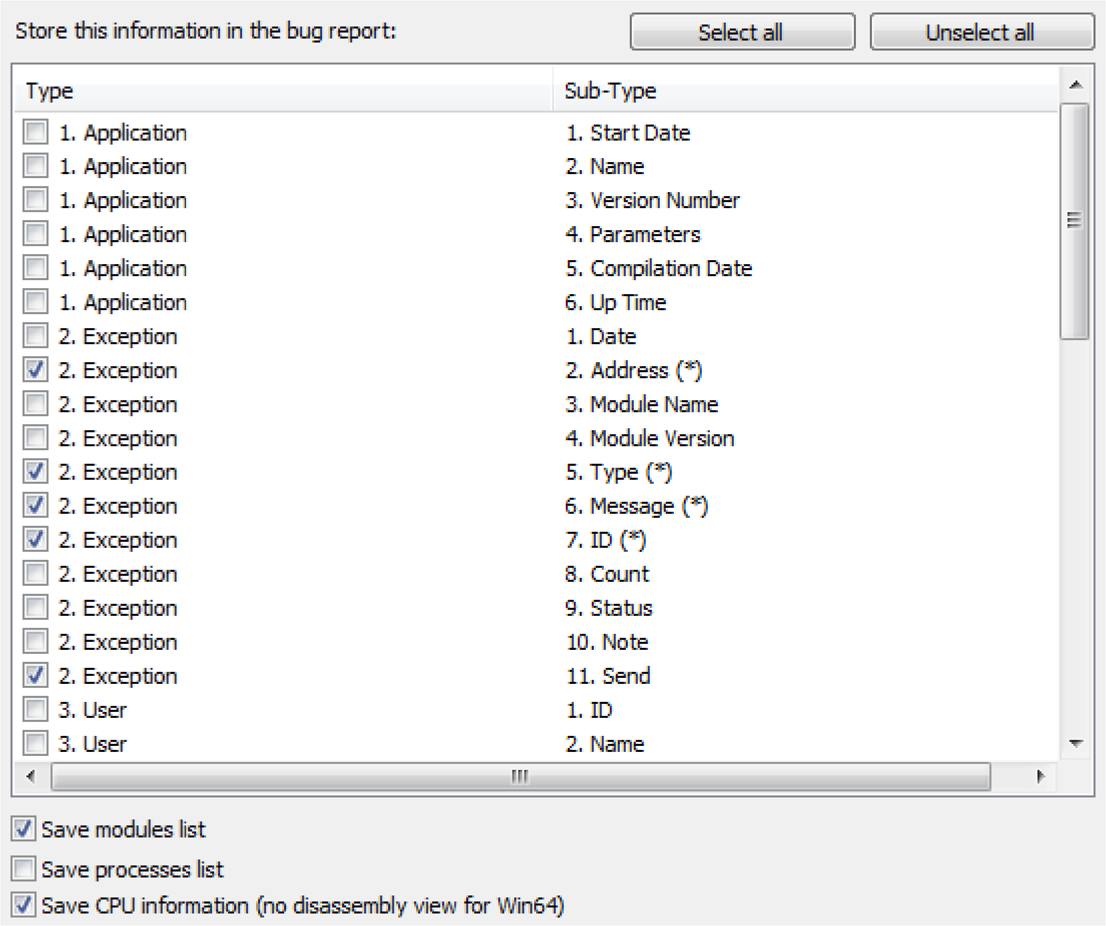
- you limited capacity of bug report's file to 1 report.
- you doesn't specify version information for your project.
- you doesn't include application version into bug report (field 1.3 266).

See also:

- Bug report content 266 page
- Configuring bug report 46
- Configuring call stack 48
- Bug reports 72

10.3.3.1 Bug report content page

This is "Bug report content" page in EurekaLog project's options 225.



Store this information in the bug report:

Type	Sub-Type
□ 1. Application	1. Start Date
□ 1. Application	2. Name
□ 1. Application	3. Version Number
□ 1. Application	4. Parameters
□ 1. Application	5. Compilation Date
□ 1. Application	6. Up Time
□ 2. Exception	1. Date
☑ 2. Exception	2. Address (*)
□ 2. Exception	3. Module Name
□ 2. Exception	4. Module Version
☑ 2. Exception	5. Type (*)
☑ 2. Exception	6. Message (*)
☑ 2. Exception	7. ID (*)
□ 2. Exception	8. Count
□ 2. Exception	9. Status
□ 2. Exception	10. Note
☑ 2. Exception	11. Send
□ 3. User	1. ID
□ 3. User	2. Name

☑ Save modules list

□ Save processes list

☑ Save CPU information (no disassembly view for Win64)

Bug report content options

Bug report content page specifies information to include into bug report. You can enable/disable information blocks here. See this article 72 for more information about bug reports.

1. "**Store this information in the bug report**" (.so*XYZ*) option controls gathering of information in general section of bug report. These are generic information pieces, which mostly describes run-time environment. Only section two ("Exception") is specific to the problem.

You command EurekaLog to gather and include into report piece of information by checking checkbox on the left from this information name. Most options can be enabled and disabled, but there are few options, which are required and can't be disabled. These options are marked with "(*)".

"Select all" and "Unselect all" buttons can be used to quickly set/clear full selection.

Warning: your application may be considered as harmful (spy-ware) by customers, if it collects extensive information which is definitely non needed (like monitor information for non-GUI applications).

2. "**Save modules list**" (`.loSaveModulesSection`) option includes information about all loaded DLLs and BPLs ("modules") into bug report.

Note: this option is automatically disabled for reporting leaks.

3. "**Save processes list**" (`.loSaveProcessesSection`) option includes information about all running processes into bug report.

Note: this option is automatically disabled for reporting leaks.

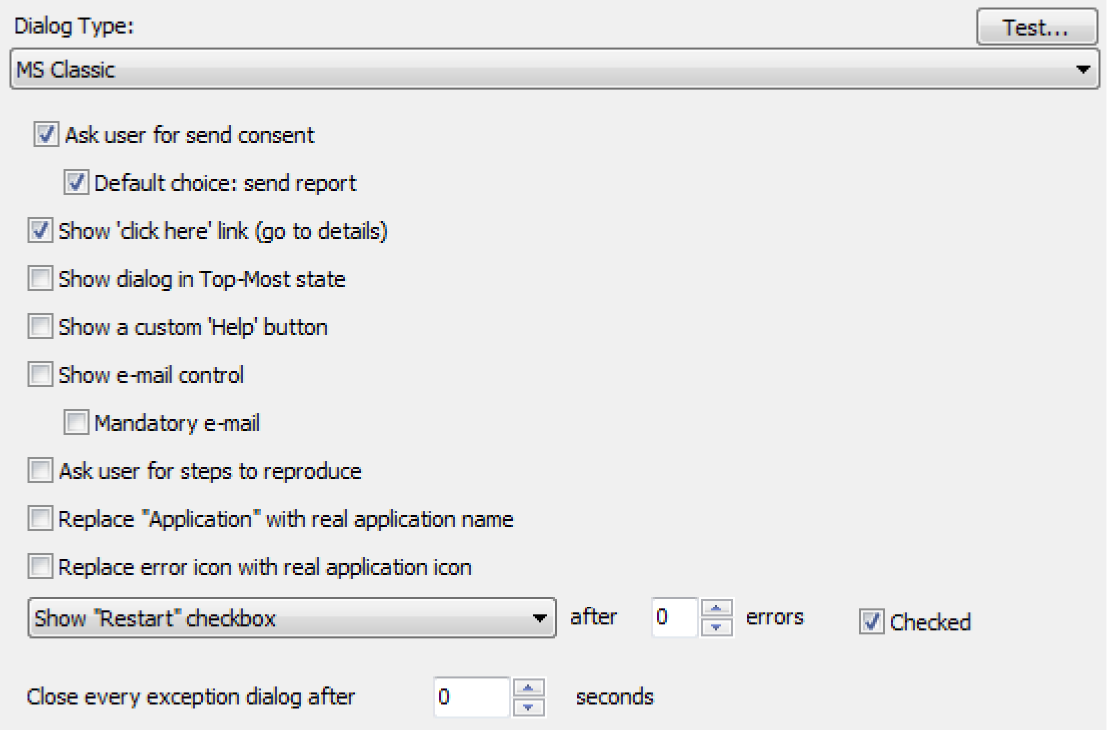
4. "**Save CPU information**" (`.loSaveAssemblerAndCPUSections`) option includes information about CPU state (flags, registers, etc) and disassembly information in bug report. This option requires installed low-level hook 349.

Note: this option is automatically disabled for reporting leaks.

See also:
- Bug report 264 options page
- Bug reports 72:
  - o General section 74
  - o Modules section 99
  - o Processes section 99
  - o Assembler section 100
  - o CPU section 103
- Configuring bug report 46
- Configuring call stack 48

10.3.4 Dialogs page

This is "Dialogs" page in EurekaLog project's options 225.



Dialog Type: [Test...]

MS Classic ▾

☑ Ask user for send consent

☑ Default choice: send report

☑ Show 'click here' link (go to details)

□ Show dialog in Top-Most state

□ Show a custom 'Help' button

□ Show e-mail control

□ Mandatory e-mail

□ Ask user for steps to reproduce

□ Replace "Application" with real application name

□ Replace error icon with real application icon

Show "Restart" checkbox ▾ after 0 errors ☑ Checked

Close every exception dialog after 0 seconds

Dialog tab with "MS Classic" dialog selected

This page contains one combo-box to select dialog type in your application. The rest of the area is changed depending on your dialog's choice.

Select appropriate dialog type for your application from "**Dialog type**" (`.ExceptionDialogType`) combo-box and specify all other options for the specific dialog type:

- (none) - have no customizable options (see: no dialog 371; `edtNone`)
- RTL - have no customizable options (see: RTL dialog 371; `edtRTL`)
- MessageBox 268
- MS Classic 271
- EurekaLog 279
- Console 292
- System log reporting 295
- WEB 296
- Windows Error Reporting 300

See also:

- Dialogs 370 to get more information about each dialog type

1. "**Test**" button allows you to test current settings. Click on this button to view how error dialog will be displayed in your application at run-time. You can use this button to quick-preview/test settings without need to close options dialog, compile your application, etc.

Note: actual exact behavior and visual aspect of dialog may be different in run-time because of your event-handlers or run-time environment. For example, you may see "Send report" button in preview, but not at run-time - because you raise duplicate exception at run-time and sending duplicates is disabled.

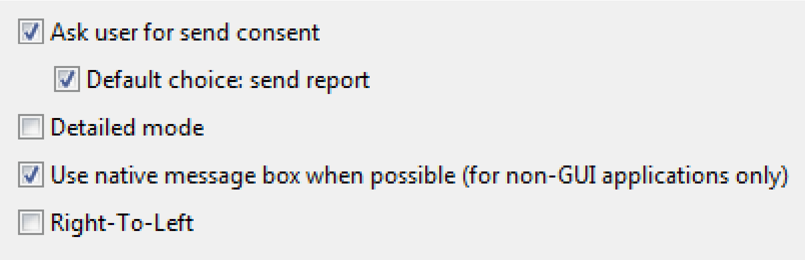
Important note: EurekaLog will automatically include code for the selected dialog into your application. However, code for other dialogs will not be included. This is important if you're going to change dialog type at run-time (via custom code 189, event handlers 192, custom attributes 190, or exception filters 185). You should manually add code for each dialog 354 that you want to use at run-time.

See also:

- Dialogs description 370
- Dialogs configuration:
  - o MessageBox 268
  - o MS Classic 271
  - o EurekaLog 279
  - o Console 292
  - o System log reporting 295
  - o WEB 296
  - o Windows Error Reporting 300
- Dialogs code configuration 354

10.3.4.1 MessageBox

This is setup options for MessageBox dialog 373 (`edtMessageBox`, `edtMessageBoxDetailed`). They are located at Dialogs tab 267.



☑ Ask user for send consent

  ☑ Default choice: send report

☐ Detailed mode

☑ Use native message box when possible (for non-GUI applications only)

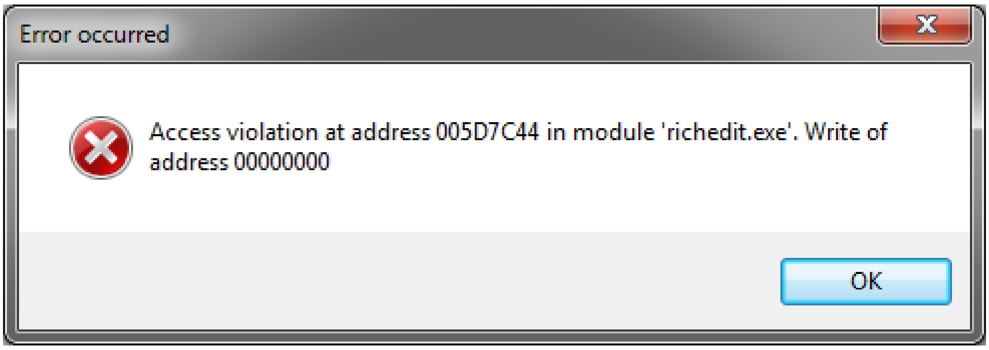
☐ Right-To-Left

MessageBox dialog options

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

1. "**Ask user for send consent**" (`.edoShowSendErrorReportOption`) option will ask user for their consent before sending bug report to developer - by showing question "Do you want to send report" and presenting "Yes" and "No" buttons.

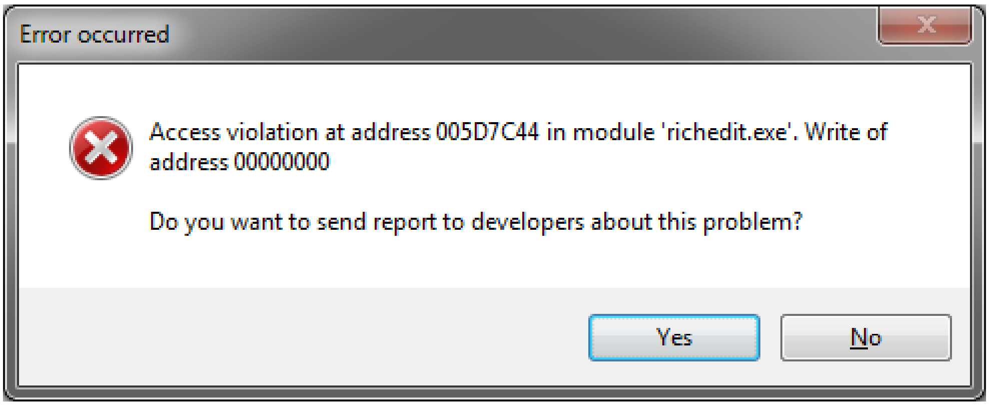
This option has no effect if you haven't specified any sending methods. In this case you'll see only one "OK" button. For example:



Error occurred

Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

Asking for consent is unchecked or there is no sending method available



Error occurred

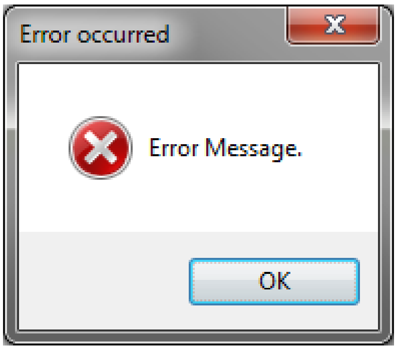
Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

Do you want to send report to developers about this problem?

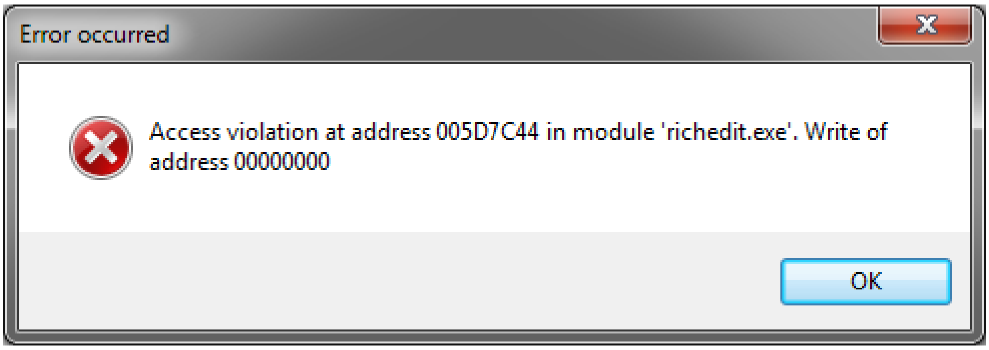
Asking for consent is checked and sending method present

2. Default choice is selected by enabling/disabling "**Default choice: send report**" (`.edoSendErrorReportChecked`) option. If this option is checked - the default choice is "send the report". If this option is unchecked - then the default choice is "do NOT send the report". Default choice affects which button (option) will be highlighted/selected when dialog is shown.

3. "**Detailed mode**" (`.ExceptionDialogType = edtMessageBoxDetailed`) option switches between standard mode and detailed mode. Standard mode shows error message only. Detailed mode shows error message and compact call stack. For example:

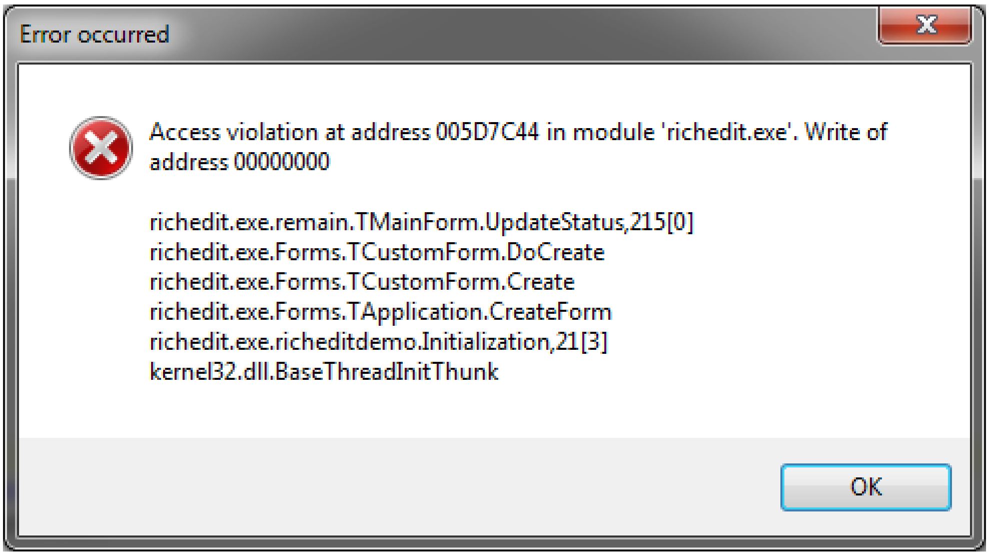


Error Message.



Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

Standard mode

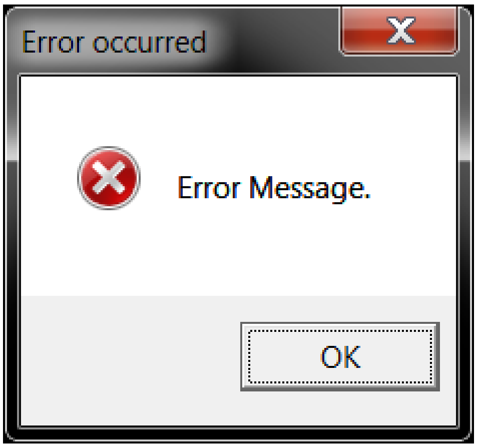


Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

```
richedit.exe.remain.TMainForm.UpdateStatus,215[0]
richedit.exe.Forms.TCustomForm.DoCreate
richedit.exe.Forms.TCustomForm.Create
richedit.exe.Forms.TApplication.CreateForm
richedit.exe.richeditdemo.Initialization,21[3]
kernel32.dll.BaseThreadInitThunk
```

Detailed mode

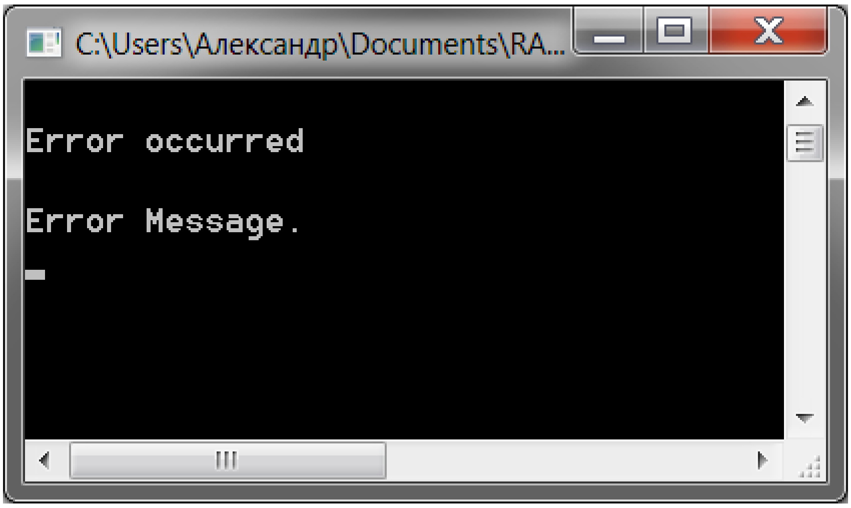
4. "**Use native message box when possible**" (`.dlgMsgBoxUseNative`) option turns on and off "native" style. By default "native" style is the same Windows.MessageBox function. However, some types of application (currently it's a console and web) overrides this to custom routines. For example, "native" message box in console application - it's an output to console. A "native" message box for IntraWeb application - it's a scripted dialog (via WebApplication.ShowMessage). For example:



Error occurred

Error Message.

"Use native message box" is off



```
Error occurred

Error Message.
```

"Use native message box" is on

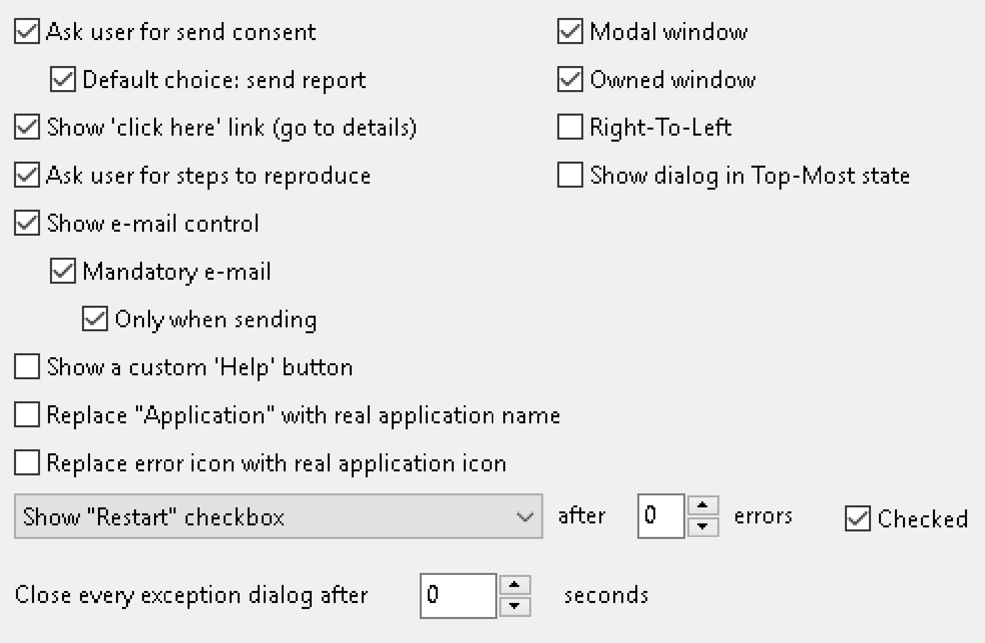
5. "**Right-To-Left**" option enables Right-To-Left layout. This is global option that affects all EurekaLog run-time dialogs. Unchecked position indicate left-to-right layout (default), checked position indicate right-to-left layout used in some middle eastern languages. This option can also be altered at design-time via Localization page 339. This option can also be altered at run-time by changing CurrentEurekaLogOptions.CustomizedTexts[mtRTLDialog].

See also:

- MessageBox dialog 373 for general description of this dialog's type
- Nested (chained) exceptions 244

10.3.4.2 MS Classic

This is setup options for MS Classic dialog 377 (edtMSClassic). They are located at Dialogs tab 267.



☑ Ask user for send consent

☑ Default choice: send report

☑ Show 'click here' link (go to details)

☑ Ask user for steps to reproduce

☑ Show e-mail control

☑ Mandatory e-mail

☑ Only when sending

□ Show a custom 'Help' button

□ Replace "Application" with real application name

□ Replace error icon with real application icon

Show "Restart" checkbox after 0 errors ☑ Checked

Close every exception dialog after 0 seconds

☑ Modal window

☑ Owned window

□ Right-To-Left

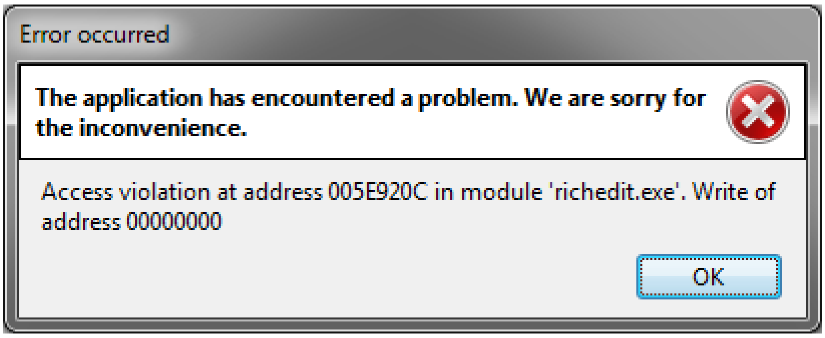
□ Show dialog in Top-Most state

MS Classic dialog options

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

1. "**Ask user for send consent**" (`.edoShowSendErrorReportOption`) option will ask user for their consent before sending bug report to developer - by showing "Please tell us about the problem" and presenting "Send Error Report" and "Don't send" buttons.

This option has no effect if you haven't specified any sending methods. In this case you'll see only one "OK" button. For example:

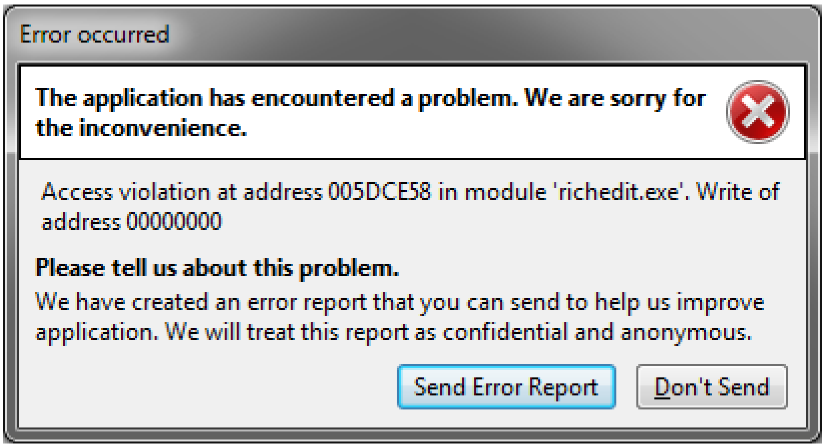


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

Asking for consent is unchecked or there is no sending method available



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

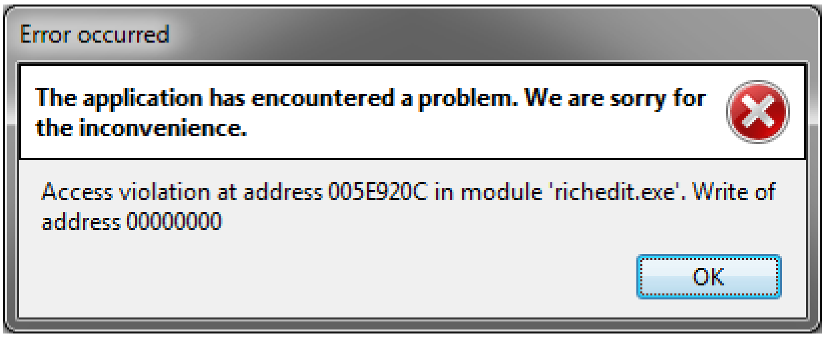
Please tell us about this problem.

We have created an error report that you can send to help us improve application. We will treat this report as confidential and anonymous.

Asking for consent is checked and sending method present

2. Default choice is selected by enabling/disabling "**Default choice: send report**" (.edoSendErrorReportChecked) option. If this option is checked - the default choice is to send the report. If this option is unchecked - then the default choice is NOT to send the report. Default choice affects which button (option) will be highlighted/selected when dialog is shown.

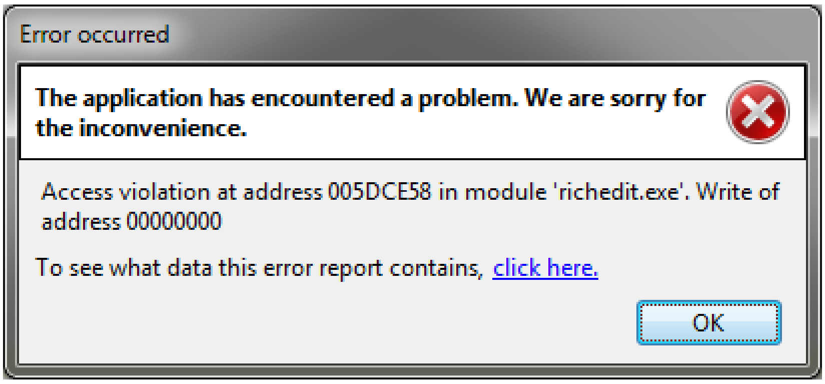
3. "**Show 'click here' link**" (.edoShowDetailsButton) option adds a "To see what data the error report contains, click here." line to the dialog. A "click here" part is a hyper-link which opens a EurekaLog style dialog 379 in "detailed" mode. For example:


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Show 'click here' link" option is unchecked


Error occurred

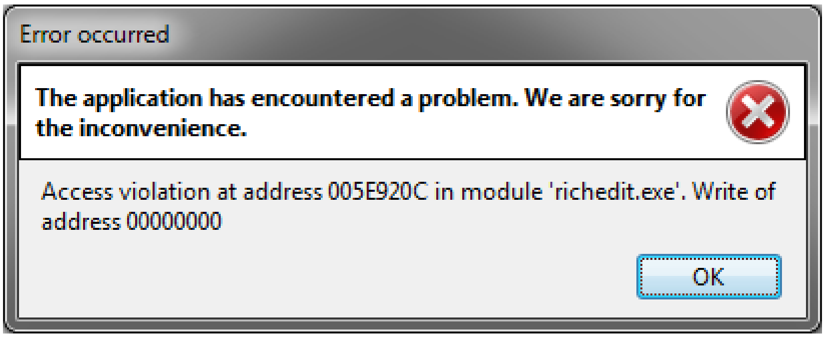
The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

To see what data this error report contains, click here.

"Show 'click here' link" option is checked

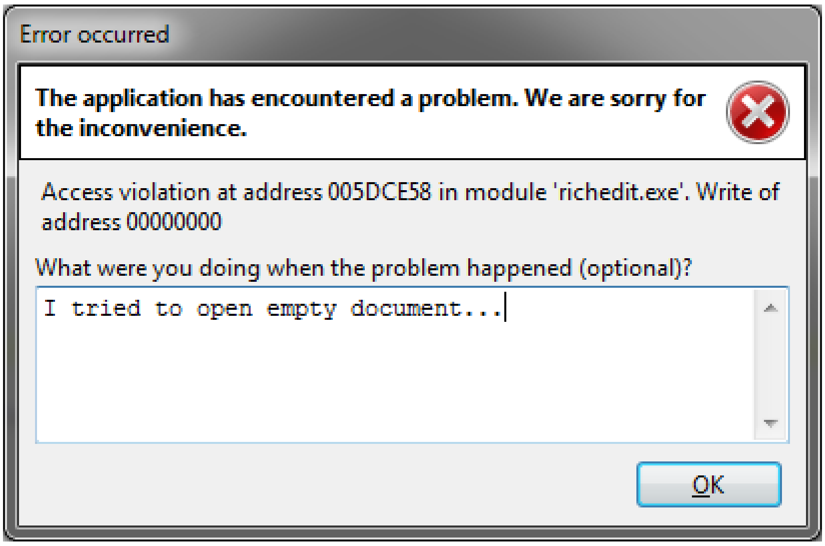
4. "**Ask user for steps to reproduce**" (.loAppendReproduceText) option shows a memo with "What were you doing when the problem happened (optional)?" header. For example:


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Ask user for steps to reproduce" option is unchecked



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

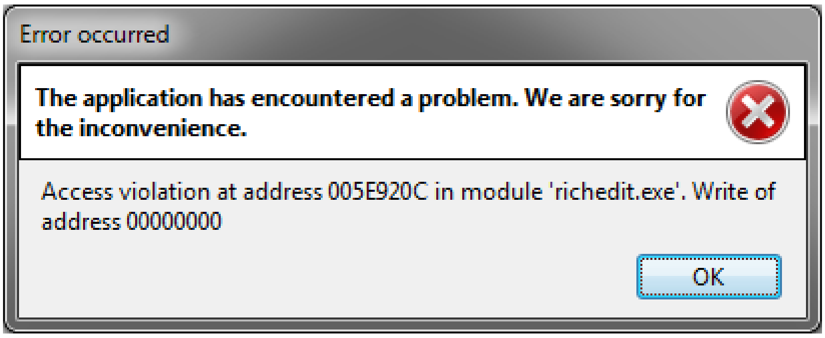
Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

What were you doing when the problem happened (optional)?

```
I tried to open empty document...
```

"Ask user for steps to reproduce" option is checked

5. "**Show e-mail control**" (`.edoShowEMailControl`) option shows the edit for user's e-mail address. You can get/set this e-mail manually by using GetUserEMail and SetUserEmail functions. For example:

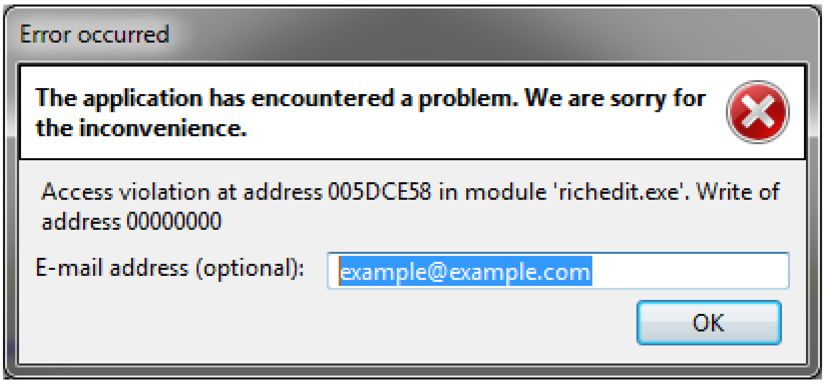


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Show e-mail control" option is unchecked



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

E-mail address (optional): example@example.com

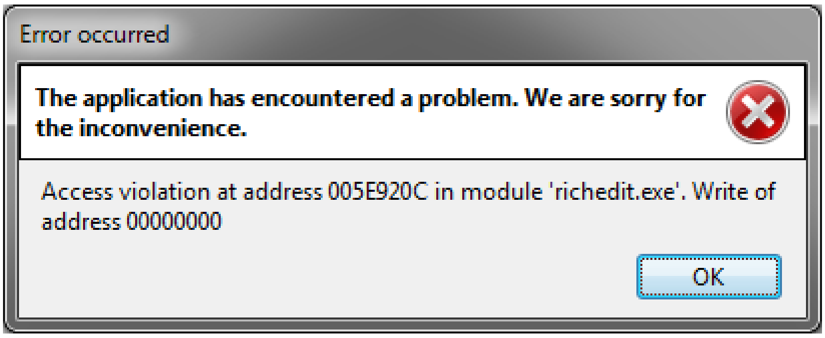
"Show e-mail control" option is checked

6. "**Mandatory e-mail**" (`.edoMandatoryEMail`) option will not allow user to close dialog without entering a proper e-mail. This option have no effect if e-mail input control is not visible. Use this option to force user to specify e-mail. This options is useful if you perform delayed/queued report sending manually. For example, you store your bug report in folder, then some application pick up/send/process this folder - and you want bug reports to have end user e-mail for contacting it.

7. "**Only when sending**" (`.edoMandatoryEMailOnlyWhenSending`) option modifies mandatory

e-mail option. When this option is disabled - e-mail will always be required. When this option is enabled - e-mail will be optional if user closes dialog without sending report. Use this option if you want user e-mail specified, and you perform sending immediately (via EurekaLog).

8. "**Show a custom 'Help' button**" (`.edoShowCustomButton`) option shows a "Help" button in the left-bottom corner of the dialog. You can assign your code on this button via the `OnCustomButtonClick` event.

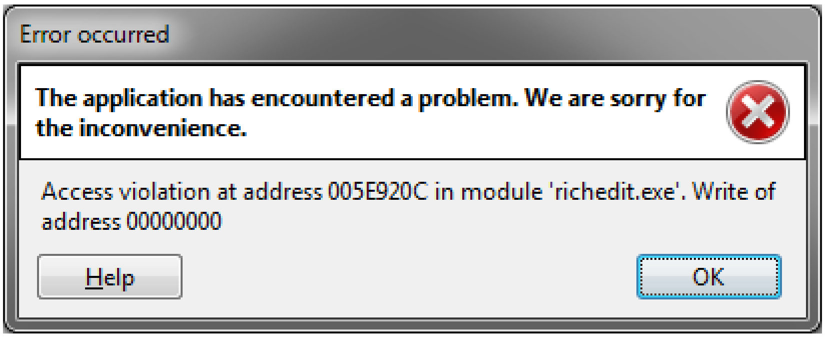


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Show a custom 'Help' button" option is unchecked



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Show a custom 'Help' button" option is checked

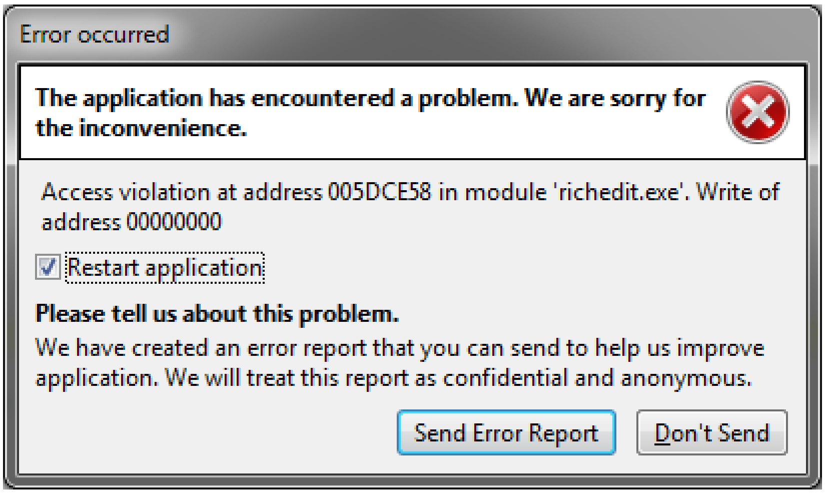
Note: while the proposed usage for this button is to act as "Help on this error" button, but you can freely implement any other behavior. You can implement arbitrary behavior by assigning OnCustomButtonClick event, and you can change caption of this button by either altering localization options 339 at design-time or assigning a new value to `CurrentEurekaLogOptions.CustomizedTexts[mtDialog_CustomButtonCaption]`.

9. "**Replace 'Application' with real application name**" (`.edoUseRealName`) option will replace "application" word in all messages on the error dialog with FileDescripton field from version information of main executable (not from module where exception has occurred) - so called "real application name" (in this article). The usual places for such replacements include:

- Window's caption is replaced to real application name.
- "The application has encountered a problem" in the header.
- "Restart application" - a checkbox to restart application.
- "Terminate application" - a checkbox to terminate application.
- "We have created an error report that you can send to help us improve application" - a description for send consent's asking.

If there is no version information available or FileDescription field is empty - then this option will have no effect and you'll see the standard "application" instead of description.

In the example below: the FileDescription field of main .exe contains 'Sample Application' string.



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

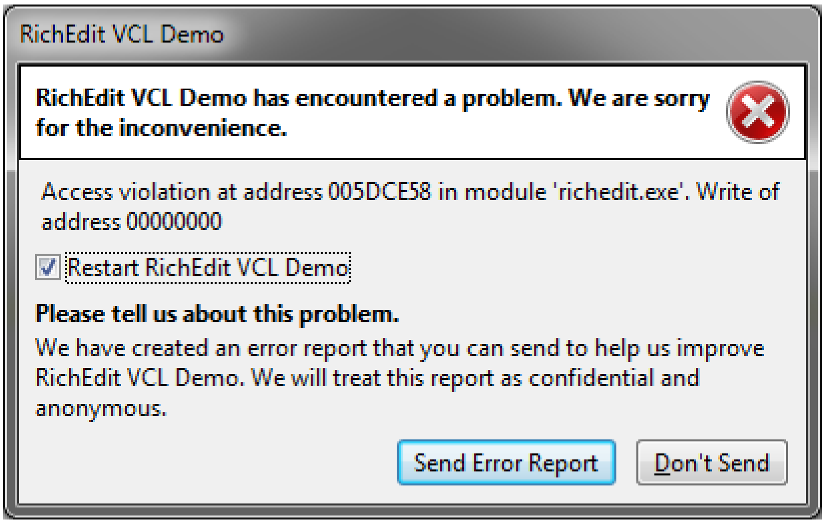
Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

☑ Restart application

Please tell us about this problem.

We have created an error report that you can send to help us improve application. We will treat this report as confidential and anonymous.

"Replace 'application' with real application name" option is unchecked



RichEdit VCL Demo

RichEdit VCL Demo has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

☑ Restart RichEdit VCL Demo

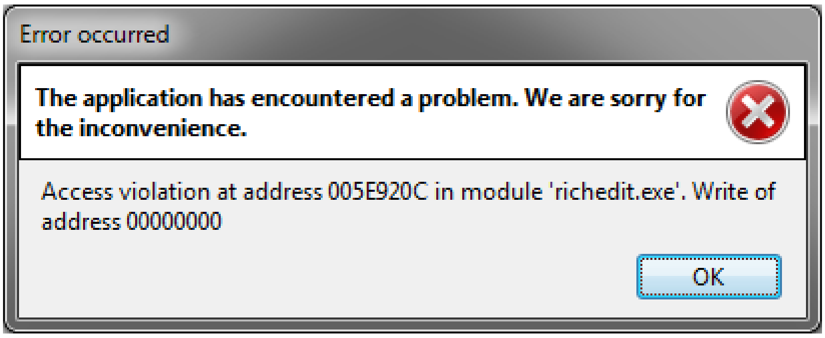
Please tell us about this problem.

We have created an error report that you can send to help us improve RichEdit VCL Demo. We will treat this report as confidential and anonymous.

"Replace 'application' with real application name" option is checked

Usually this option is used together with "Replace error icon with real application icon" option (see below) to get a fully personalized view.

10. "**Replace error icon with real application icon**" (`.edoUseRealIcon`) option will replace standard IDI_ERROR icon to icon of the main .exe (i.e. the first icon). For example:

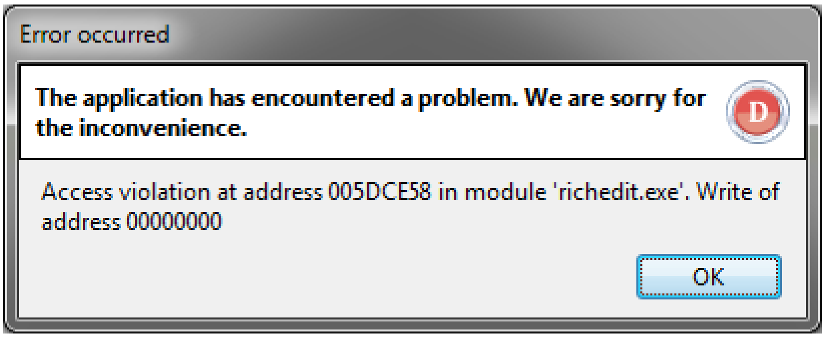


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Replace error icon with real application icon" option is unchecked


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

"Replace error icon with real application icon" option is checked

Usually this option is used together with "Replace 'application' with real application name" option (see above) to get a fully personalized view.

11. "**Modal window**" (.edoShowModal) option makes error dialog modal. Modal means that only error dialog will be accessible. All other windows in current thread will be disabled (user will not be able to interact with them). Windows from other processes and from other threads within the same process will be accessible. When this option is unchecked - no other windows will be disabled.

Note: usually it's not good idea to uncheck "**Owner window**" option when "**Modal window**" option is checked.

12. "**Owner window**" (.edoOwnedWindow) option makes error dialog an owned window to currently active window (owner window). Here: Owner-Owned relation is used in system's meaning (as relation between two HWND) as opposed to Delphi's meaning (as relation between two TComponent).

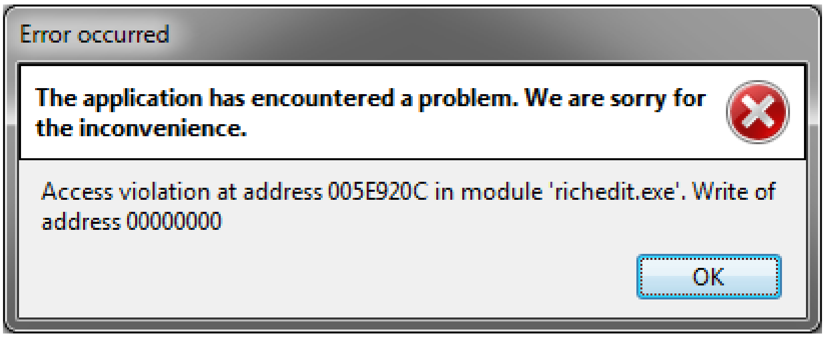
Checked: error dialog will be displayed as owned window to currently active window. If there is no active window - this option will have no effect. Being owned places several constraints on a window. Generally, owner-owned windows act as group. For example, an owned window is always above its owner in the z-order. Both owner and owned windows "share" the same button on taskbar (actually, owned window do not have taskbar button). An owned window is hidden when its owner is minimized.
Unchecked: error dialog will not be related to any other window. It will be standalone window.

Note: usually it's not good idea to uncheck "**Owner window**" option when "**Modal window**" option is checked.

13. "**Right-To-Left**" option enables Right-To-Left layout. This is global option that affects all EurekaLog run-time dialogs. Unchecked position indicate left-to-right layout (default), checked position indicate right-to-left layout used in some middle eastern languages. This option can also be altered at design-time via Localization page 339. This option can also be altered at run-time by changing CurrentEurekaLogOptions.CustomizedTexts[mtRTLDialog].

14. "**Show dialog in Top-Most state**" (.edoShowInTopMostMode) option sets the HWND_TOPMOST position to error dialog. If this option is checked then error window will appear above all other non-TopMost windows. If this option is unchecked then HWND_TOPMOST will not be set. Instead, a timer is run. This timer will call SetForegroundWindow for error dialog, if it was covered by any other process's windows. Windows from other processes can cover error window. Windows from current process, which are created after error dialog can cover error dialog too. This timer doesn't work in TopMost mode.

15. **"Do nothing" / "Show 'Restart' checkbox" / "Show 'Terminate' checkbox" after N errors** (.TerminateBtnOperation, .ErrorsNumberToShowTerminateBtn, .edoRestartChecked) option controls the visibility of "restart" and "terminate" checkboxes:

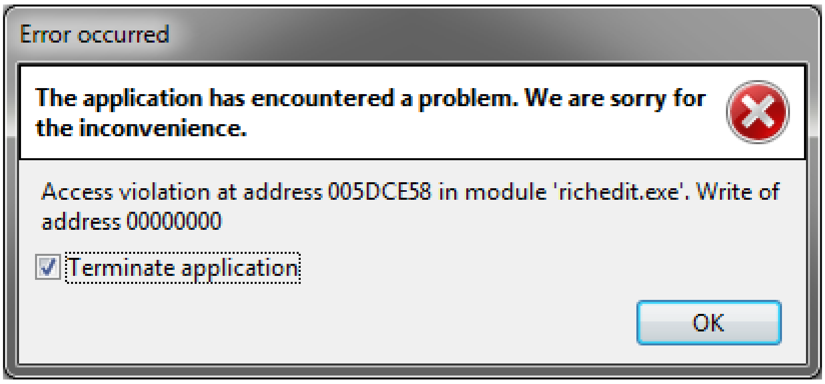


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

"Do nothing" option is selected or error count is less than the specified count.



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005DCE58 in module 'richedit.exe'. Write of address 00000000

☑ Terminate application

"Show 'Terminate' checkbox" options is selected and error count is more than the specified count.

The "Checked" option controls if this checkbox is initially checked or unchecked. If "Checked" option is checked, then when MS Classic dialog appears with restart/terminate checkbox visible - this checkbox will be checked. And the opposite: if the "Checked" option is not checked, then restart/terminate checkbox will be unchecked, when dialog appears. Of course, this option have no effect, if checkbox is not visible (for example, if you select "Do nothing" option).

A "after N errors" part is controls, when to show restart/terminate checkbox. Again, this part is ignored, if you select "Do nothing" option. Value of 0 means that restart/terminate checkbox will be visible always. Value of 1 means that checkbox will be displayed only in second error dialog. Value of 2 means that you got two error dialog without checkbox and 3rd dialog will have it. And so on.

At run-time: when user checks restart/terminate checkbox, then your application will be restarted or terminated after closing error dialog and saving and sending bug report (if enabled). If user unchecks the checkbox - then your application will continue to run.

16. "**Close every exception dialog after M seconds**" (`.AutoCloseDialogSecs`) option allows you to automatically close exception dialogs after timeout of inactivity. Value of 0 means disabling of such feature - i.e. each dialog may be closed only manually by user (via clicking on dialog's buttons). Any other value (> 0) means that dialog will be closed after that amount of seconds, passed since its popup. For example, if you specify 180 seconds - then dialog will be closed exactly after 3 minutes, if user didn't closed it before.

This option is useful to auto-close error dialogs in possible non-interactive scenarios. Note, that it may be preferable to select other dialog type (non-GUI) in such cases.

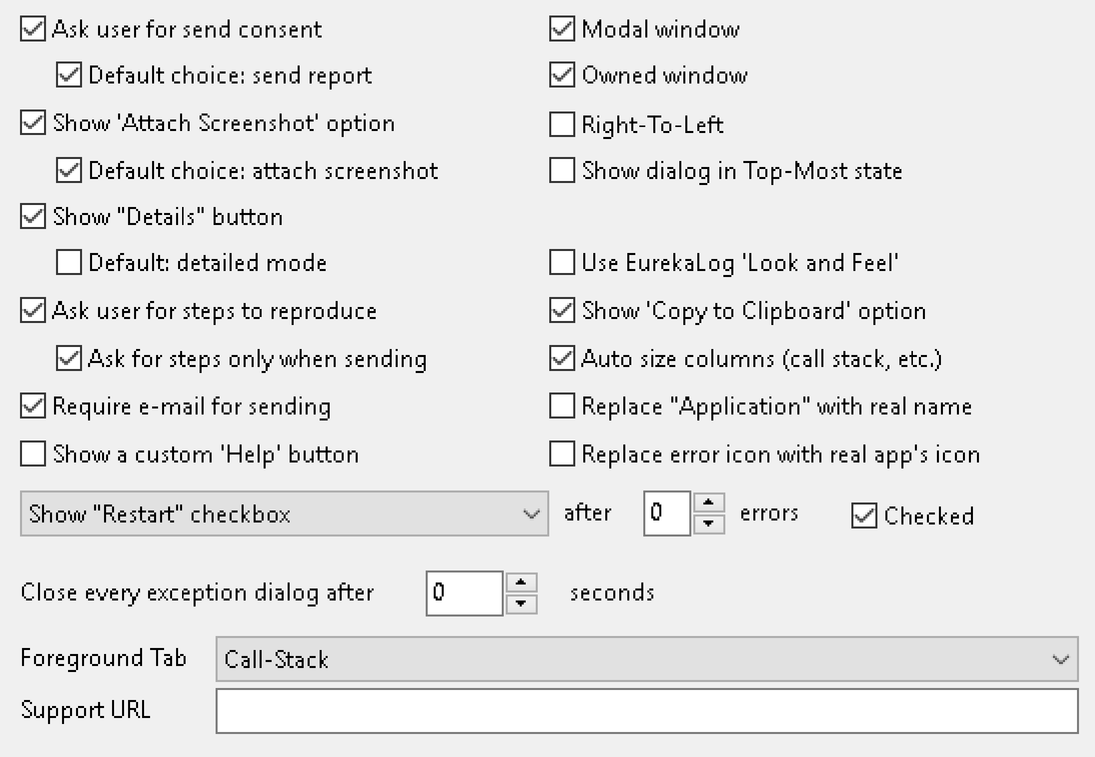
Note: this option specifies the delay of user's **inactivity**. If user moves mouse, presses buttons or generates any other input in dialog - auto-close timer will reset. Therefore, setting this option to, say, 5 seconds does not mean that dialog will be closed after 5 seconds. It may stay for, say, 1 minute - but only if user is working with it. Dialog will be closed after 5 seconds if user is away (not moving mouse, etc.).

See also:

- MS Classic dialog 377 for general description of this dialog's type
- Nested (chained) exceptions 244

10.3.4.3 EurekaLog

This is setup options for EurekaLog dialog 379 (`edtEurekaLog`, `edtEurekaLogDetailed`). They are located at Dialogs tab 267.



☑ Ask user for send consent
☑ Default choice: send report
☑ Show 'Attach Screenshot' option
☑ Default choice: attach screenshot
☑ Show "Details" button
□ Default: detailed mode
☑ Ask user for steps to reproduce
☑ Ask for steps only when sending
☑ Require e-mail for sending
□ Show a custom 'Help' button

☑ Modal window
☑ Owned window
□ Right-To-Left
□ Show dialog in Top-Most state
□ Use EurekaLog 'Look and Feel'
☑ Show 'Copy to Clipboard' option
☑ Auto size columns (call stack, etc.)
□ Replace "Application" with real name
□ Replace error icon with real app's icon

Show "Restart" checkbox after 0 errors ☑ Checked

Close every exception dialog after 0 seconds

Foreground Tab: Call-Stack

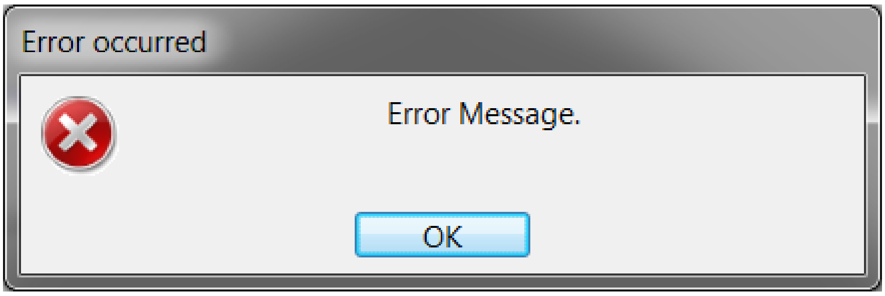
Support URL:

EurekaLog dialog options

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

1. "**Ask user for send consent**" (`.edoShowSendErrorReportOption`) option will ask user for their consent before sending bug report to developer - by showing "Send this error via Internet" checkbox.

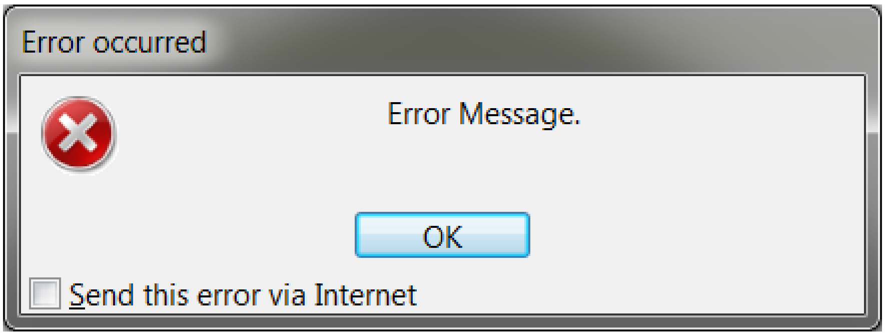
This option has no effect if you haven't specified any sending methods. In this case you'll see only one "OK" button. For example:



Error occurred

Error Message.

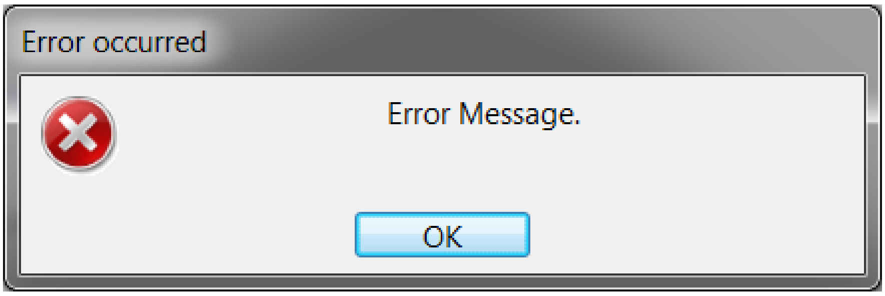
Asking for consent is unchecked or there is no sending method available



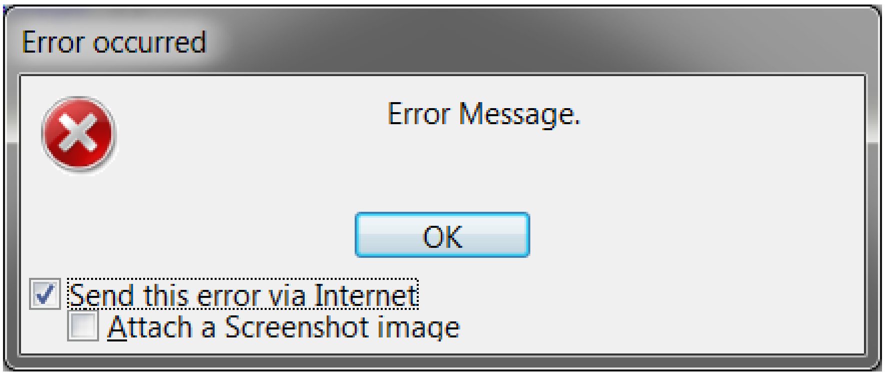
Asking for consent is checked and sending method present

2. A default choice (checked or cleared) is selected by enabling/disabling "**Default choice: send report**" (.edoSendErrorReportChecked) option. If this option is checked - the default choice is to send the report. If this option is unchecked - then the default choice is NOT to send the report. This affects the initial state of "Send this error via Internet" checkbox.

3. "**Show 'Attach screenshot' option**" (`.edoShowAttachScreenshotOption`) option will add the "Attach a screenshot image" checkbox to the dialog. This option has no effect if "Ask user for send consent" option is not checked, the sending method is not selected or getting screenshots is disabled from send options 304.



Show attach screenshot option is unchecked or other conditions aren't hold

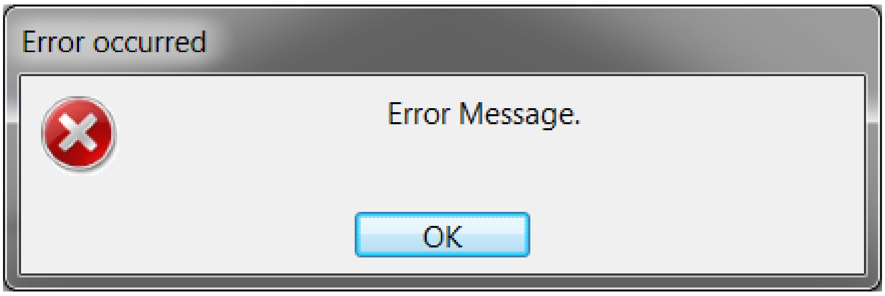


Show attach screenshot option is checked and other conditions were completed

This checkbox controls whenever a screenshot in included into bug report for send or not.

4. A default state (checked or cleared) for "Attach a screenshot image" checkbox is selected via "**Default choice: attach screenshot**" (`.edoAttachScreenshotChecked`) option.

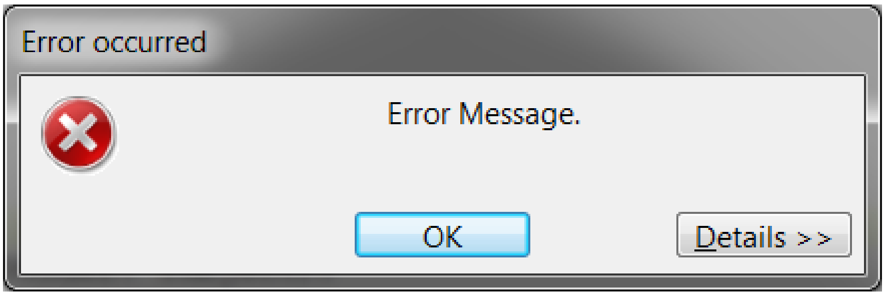
5. "**Show 'Details' button**" (`.edoShowDetailsButton`) option will show a "Details" button which switches dialog to detailed mode. If dialog is already in detailed mode, then this button will switch dialog back to non-detailed mode.



Error occurred

Error Message.

"Show 'Details' button" option is unchecked

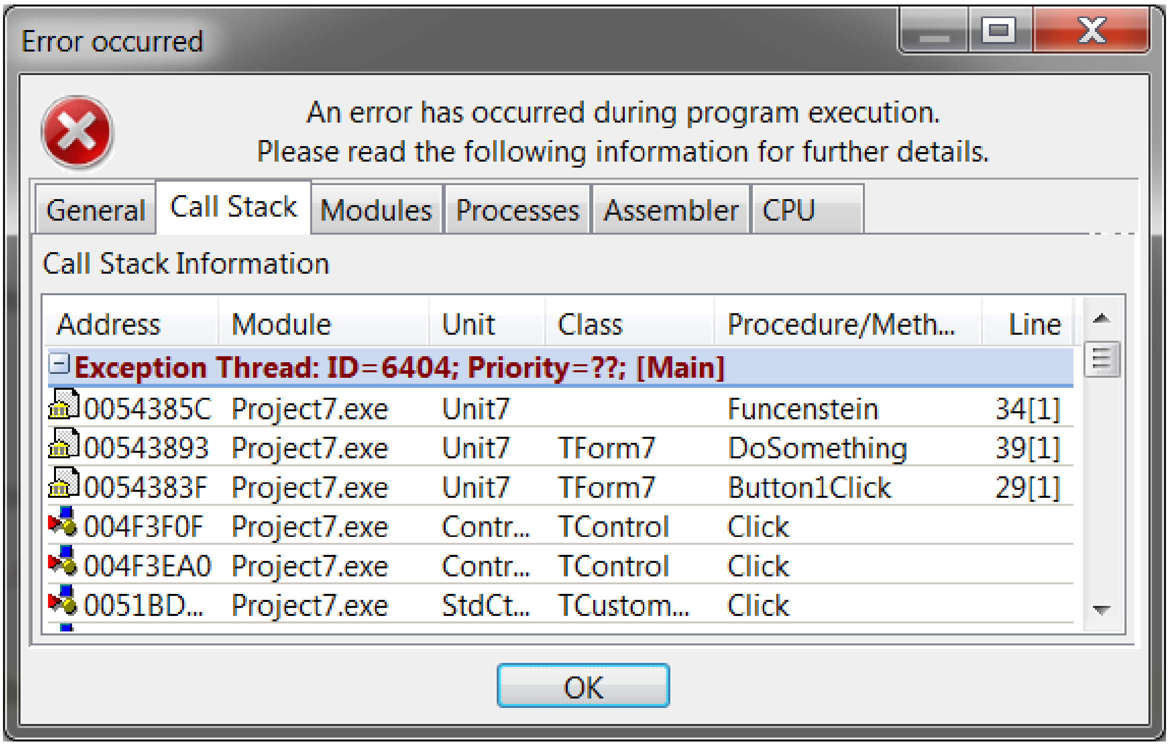


Error occurred

Error Message.

"Show 'Details' button" option is checked

Detailed mode shows full bug report:



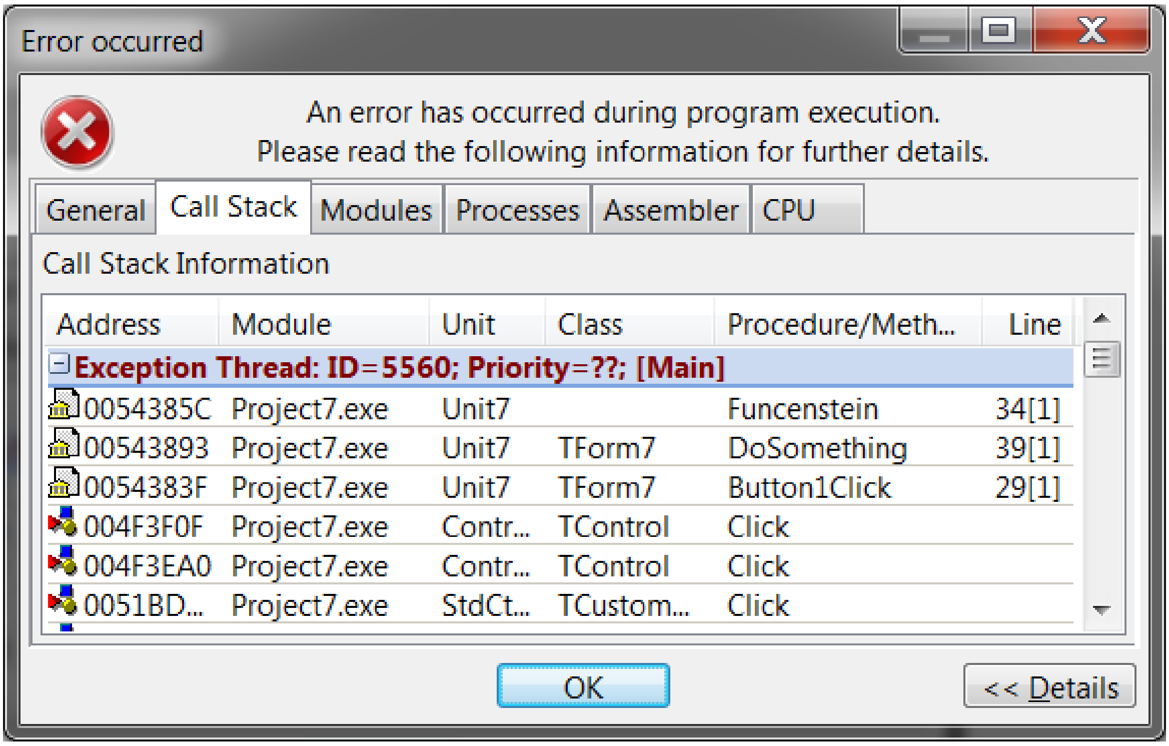
Error occurred

An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
⊟ Exception Thread: ID=6404; Priority=??; [Main]					
0054385C	Project7.exe	Unit7		Funcenstein	34[1]
00543893	Project7.exe	Unit7	TForm7	DoSomething	39[1]
0054383F	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F3F0F	Project7.exe	Contr...	TControl	Click	
004F3EA0	Project7.exe	Contr...	TControl	Click	
0051BD...	Project7.exe	StdCt...	TCustom...	Click	

EurekaLog dialog in detailed mode with "Show 'Details' button" option unchecked



An error has occurred during program execution.
Please read the following information for further details.

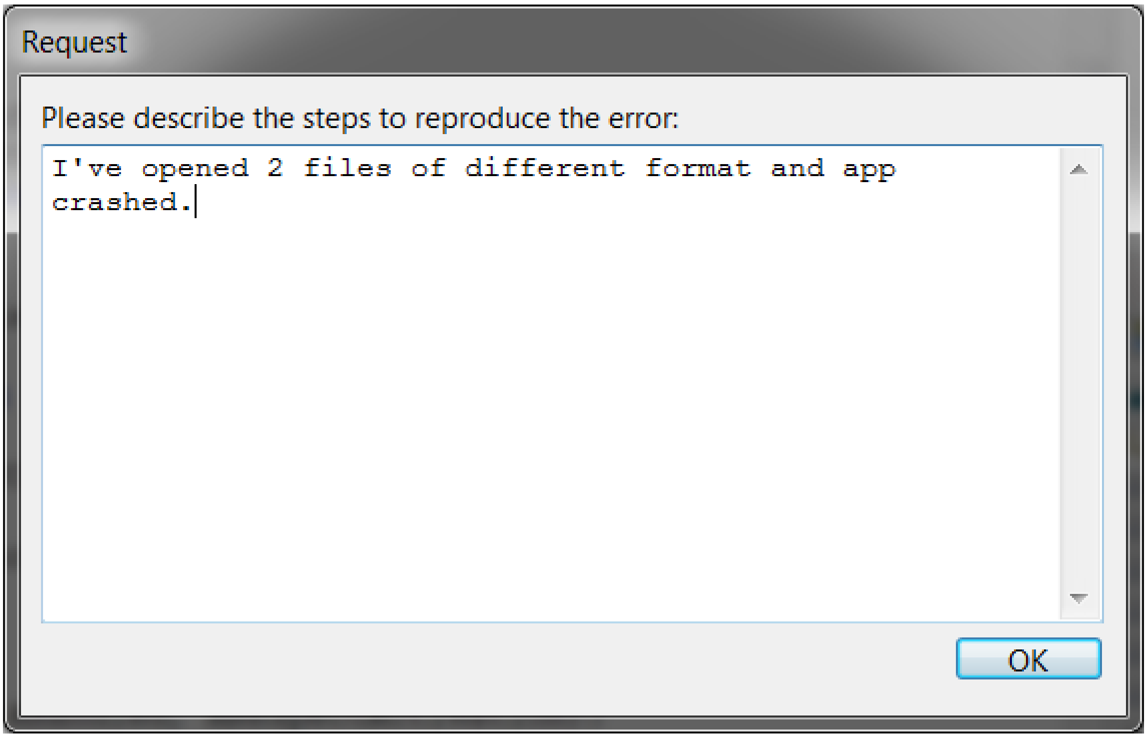
Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
Exception Thread: ID=5560; Priority=??; [Main]					
0054385C	Project7.exe	Unit7		Funcenstein	34[1]
00543893	Project7.exe	Unit7	TForm7	DoSomething	39[1]
0054383F	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F3F0F	Project7.exe	Contr...	TControl	Click	
004F3EA0	Project7.exe	Contr...	TControl	Click	
0051BD...	Project7.exe	StdCt...	TCustom...	Click	

EurekaLog dialog in detailed mode with "Show 'Details' button" option checked

6. You can control whenever dialog is initially showed in standard or detailed mode by switching "**Default: detailed mode**" (.ExceptionDialogType = edtEurekaLogDetailed) option.

7. "**Ask for steps to reproduce**" (.loAppendReproduceText) option will ask user for additional description of the problem in a separate window:



Please describe the steps to reproduce the error:

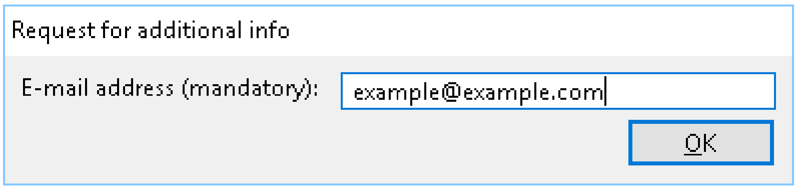
```
I've opened 2 files of different format and app
crashed.
```

Standalone window asking for "step to reproduce"

This window appears after closing of EurekaLog error dialog. After user clicks on OK button, EurekaLog will perform usual processing (saving and sending), which depends on options selected in original EurekaLog style dialog.

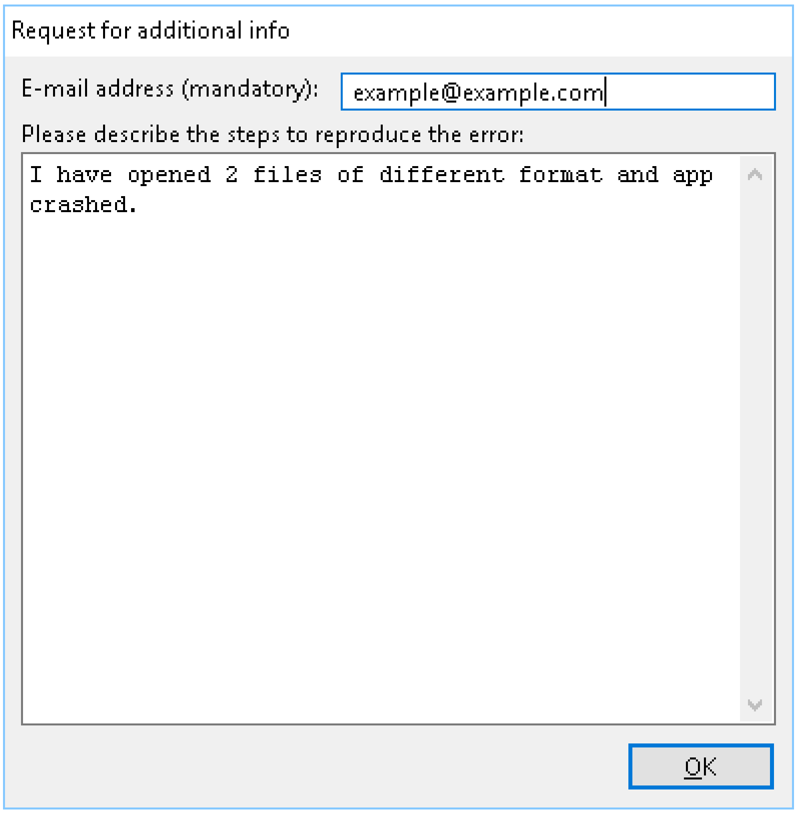
8. "**Ask for steps only when sending**" (`.loAppendReproduceTextOnlyWhenSending`) option changes behaviour or "Ask for steps to reproduce" option. It has no effect if "Ask for steps to reproduce" option is unchecked. If "Ask for steps to reproduce" option is checked, then you can disable showing "steps to reproduce" window when "Send report via Internet" checkbox is unchecked at run-time - by checking "Ask for steps only when sending" option. In other words, "Ask for steps only when sending" option is unchecked: ask always. "Ask for steps only when sending" option is checked: ask only when "Send report via Internet" is checked too.

9. "**Require e-mail for sending**" (`.edoShowEMailControl`, `.edoMandatoryEMail`, `.edoMandatoryEMailOnlyWhenSending`) option will additionally ask user's e-mail before sending the bug report. This option have no effect if sending is disabled or if a user has selected NOT to send a bug report.


Request for additional info

E-mail address (mandatory): example@example.com

Standalone window asking for user's e-mail


Request for additional info

E-mail address (mandatory): example@example.com

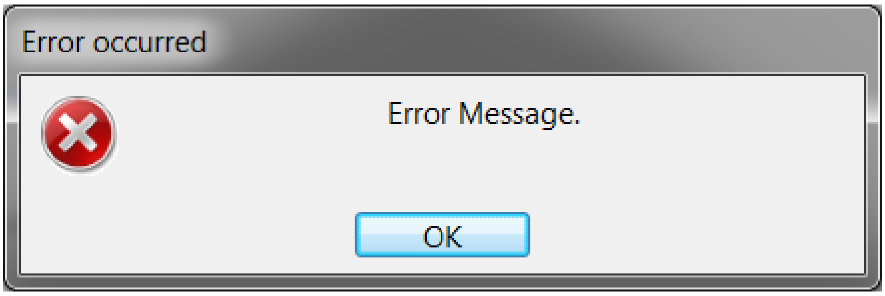
Please describe the steps to reproduce the error:

I have opened 2 files of different format and app crashed.

Same window if both "Require e-mail for sending" and "Ask user for steps to reproduce" options are enabled

Entering e-mail is mandatory. This dialog will not appear if e-mail was already specified earlier - for example, in previous error dialog, or e-mail was entered in another EurekaLog-enabled application or even another instance of the same application (EurekaLog saves user's e-mail in registry, you can work with it via GetUserEMail and SetUserEmail functions).

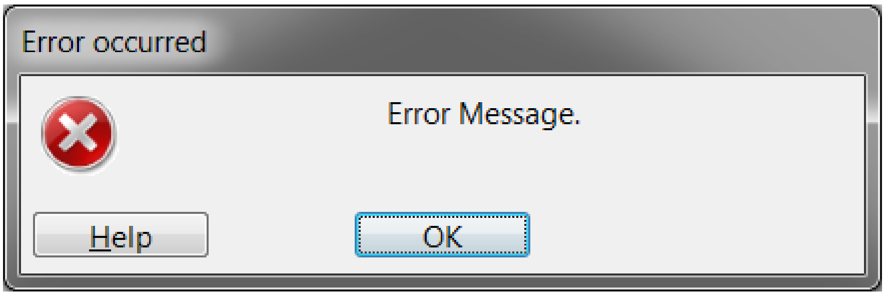
10. "**Show a custom 'Help' button**" (`.edoShowCustomButton`) option shows a "Help" button in the left-bottom corner of the dialog. You can assign your code on this button via the OnCustomButtonClick event.



Error occurred

Error Message.

"Show a custom 'Help' button" option is unchecked



Error occurred

Error Message.

"Show a custom 'Help' button" option is checked

Note: while the proposed usage for this button is to act as "Help on this error" button, but you can freely implement any other behavior. You can implement arbitrary behavior by assigning OnCustomButtonClick event, and you can change caption of this button by either altering localization options 339 at design-time or assigning a new value to `CurrentEurekaLogOptions.CustomizedTexts[mtDialog_CustomButtonCaption]`.

11. "**Modal window**" (`.edoShowModal`) option makes error dialog modal. Modal means that only error dialog will be accessible. All other windows in current thread will be disabled (user will not be able to interact with them). Windows from other processes and from other threads within the same process will be accessible. When this option is unchecked - no other windows will be disabled.

Note: usually it's not good idea to uncheck "**Owner window**" option when "**Modal window**" option is checked.

12. "**Owner window**" (`.edoOwnedWindow`) option makes error dialog an owned window to currently active window (owner window). Here: Owner-Owned relation is used in system's meaning (as relation between two HWND) as opposed to Delphi's meaning (as relation between two `TComponent`).

Checked: error dialog will be displayed as owned window to currently active window. If there is no active window - this option will have no effect. Being owned places several constraints on a window. Generally, owner-owned windows act as group. For example, an owned window is always above its owner in the z-order. Both owner and owned windows "share" the same button on taskbar (actually, owned window do not have taskbar button). An owned window is hidden when its owner is minimized.
Unchecked: error dialog will not be related to any other window. It will be standalone window.

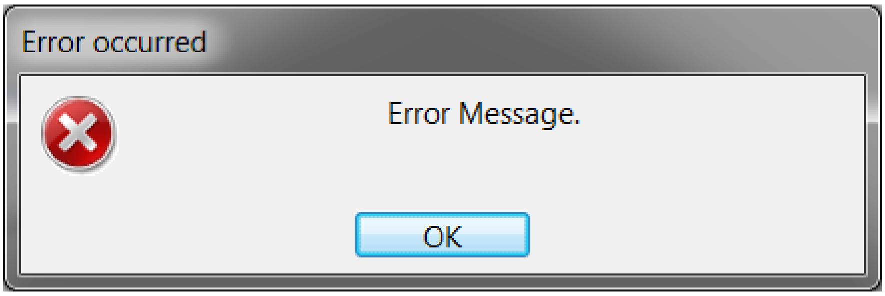
Note: usually it's not good idea to uncheck "**Owner window**" option when "**Modal window**" option is checked.

13. "**Right-To-Left**" option enables Right-To-Left layout. This is global option that affects all EurekaLog run-time dialogs. Unchecked position indicate left-to-right layout (default), checked position indicate right-to-left layout used in some middle eastern languages. This

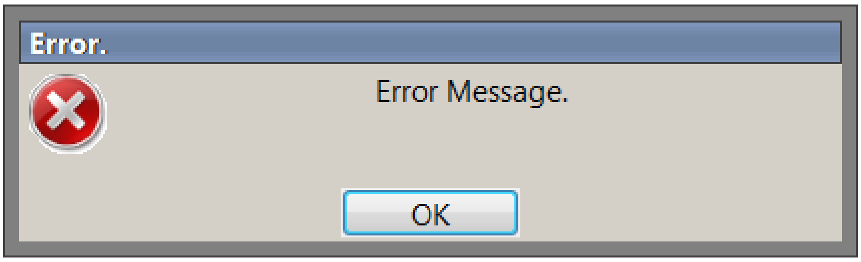
option can also be altered at design-time via Localization page 339. This option can also be altered at run-time by changing CurrentEurekaLogOptions.CustomizedTexts[mtRTLDialog].

14. "**Show dialog in Top-Most state**" (`.edoShowInTopMostMode`) option sets the HWND_TOPMOST position to error dialog. If this option is checked then error window will appear above all other non-TopMost windows. If this option is unchecked then HWND_TOPMOST will not be set. Instead, a timer is run. This timer will call SetForegroundWindow for error dialog, if it was covered by any other process's windows. Windows from other processes can cover error window. Windows from current process, which are created after error dialog can cover error dialog too. This timer doesn't work in TopMost mode.

15. "**Use EurekaLog 'look and feel'**" (`.edoUseEurekaLogLookAndFeel`) option allows you to change visual dialog style like this:

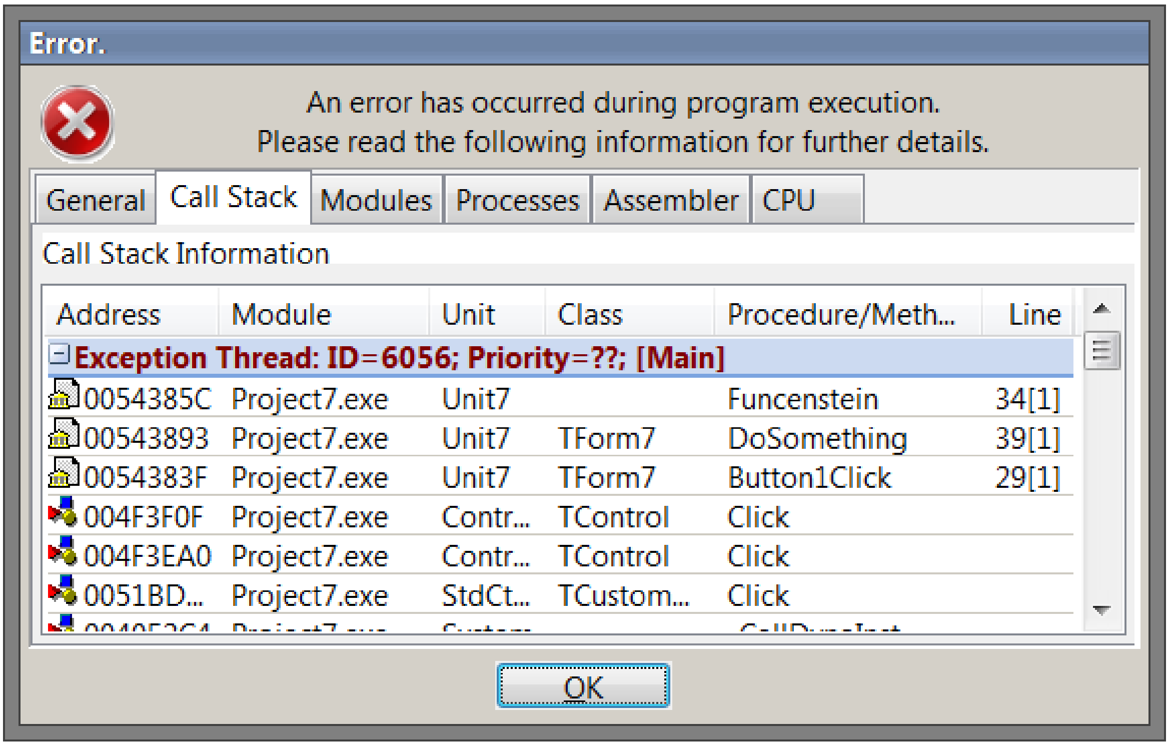


"Use EurekaLog 'look and feel'" option is unchecked



"Use EurekaLog 'look and feel'" option is checked

This allows you to visually get attention to your error dialogs. This modification works for detailed mode too:



Error.

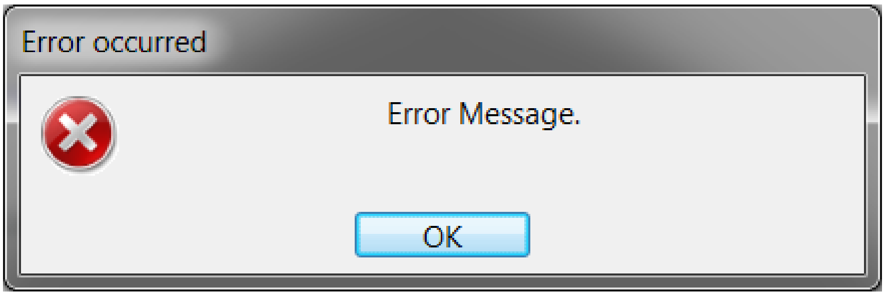
An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
Exception Thread: ID=6056; Priority=??; [Main]					
0054385C	Project7.exe	Unit7		Funcenstein	34[1]
00543893	Project7.exe	Unit7	TForm7	DoSomething	39[1]
0054383F	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F3F0F	Project7.exe	Contr...	TControl	Click	
004F3EA0	Project7.exe	Contr...	TControl	Click	
0051BD...	Project7.exe	StdCt...	TCustom...	Click	

Both "Use EurekaLog 'look and feel'" and "Default: detailed mode" options are checked

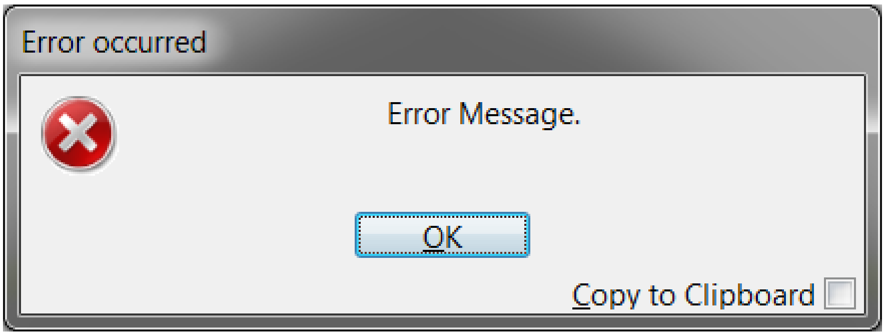
16. "**Show 'Copy to clipboard' option**" (.edoShowCopyToClipOption) option shows a "Copy to clipboard" checkbox.



Error occurred

Error Message.

"Show 'Copy to clipboard' button" option is unchecked



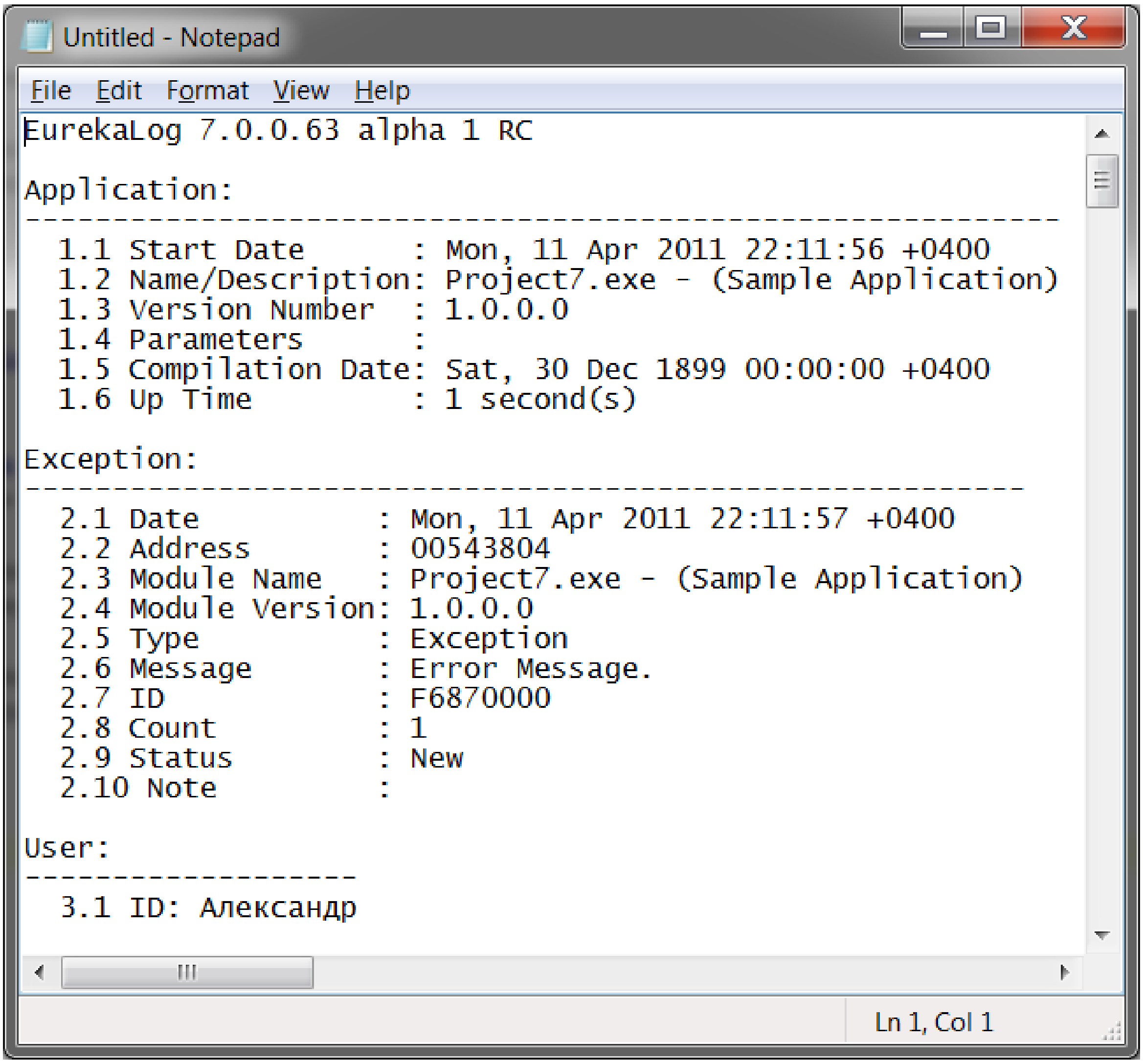
Error occurred

Error Message.

"Show 'Copy to clipboard' button" option is checked

At run-time: user can check "Copy to clipboard" checkbox to get full bug report to be copied into clipboard, so he can insert it into a e-mail or a text file manually.

For example:



```
EurekaLog 7.0.0.63 alpha 1 RC

Application:
-----------------------------------------------------------
  1.1 Start Date      : Mon, 11 Apr 2011 22:11:56 +0400
  1.2 Name/Description: Project7.exe - (Sample Application)
  1.3 Version Number  : 1.0.0.0
  1.4 Parameters      :
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0400
  1.6 Up Time         : 1 second(s)

Exception:
---------------------------------------------------------
  2.1 Date          : Mon, 11 Apr 2011 22:11:57 +0400
  2.2 Address       : 00543804
  2.3 Module Name   : Project7.exe - (Sample Application)
  2.4 Module Version: 1.0.0.0
  2.5 Type          : Exception
  2.6 Message       : Error Message.
  2.7 ID            : F6870000
  2.8 Count         : 1
  2.9 Status        : New
  2.10 Note         :

User:
-------------------
  3.1 ID: Александр
```

A Windows Notepad with bug report, inserted from clipboard

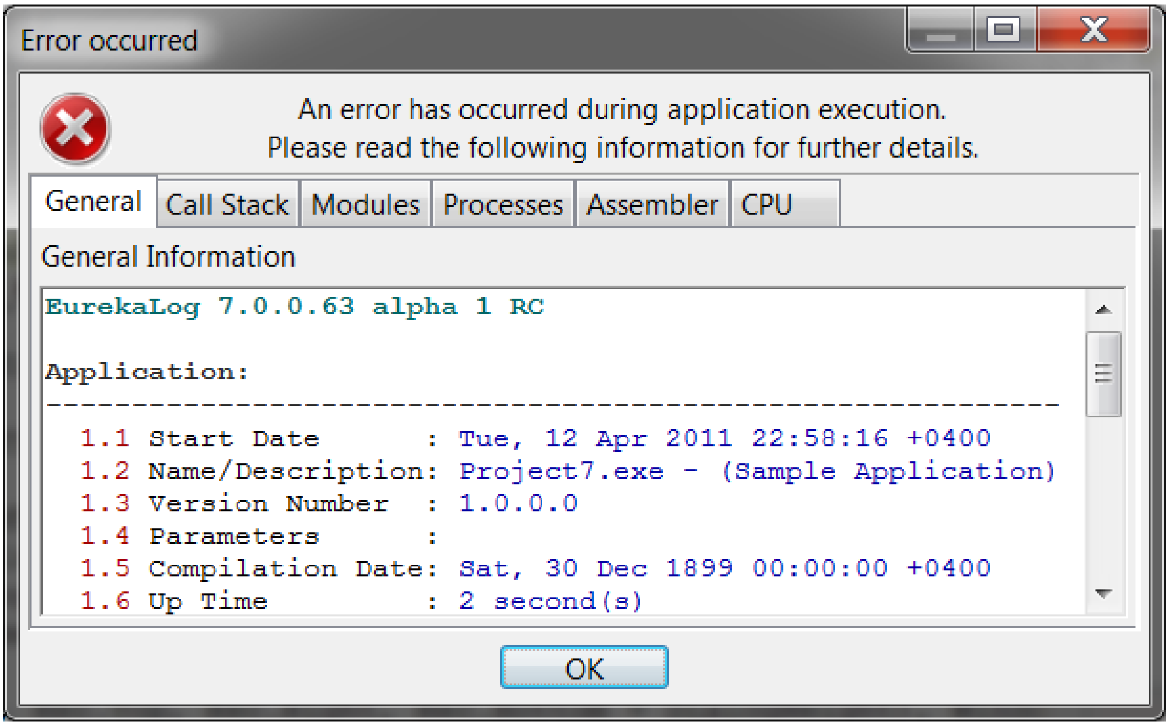
17. "**Auto-size columns**" (.edoAutoSize) option will automatically resize columns to fit column's content. User still be able to resize column manually. If this option is disabled - columns will not be resized, columns will keep previously saved widths.

18. "**Replace 'Application' with real application name**" (.edoUseRealName) option will replace "application" word in all messages on the error dialog with FileDescripton field from version information of main executable (not from module where exception has occurred) - so called "real application name" (in this article). The usual places for such replacements include:

- Window's caption is replaced to real application name.
- "An error has occurred during application execution" in the header in detailed mode.

If there is no version information available or FileDescription field is empty - then this option will have no effect and you'll see the standard "application" instead of description.

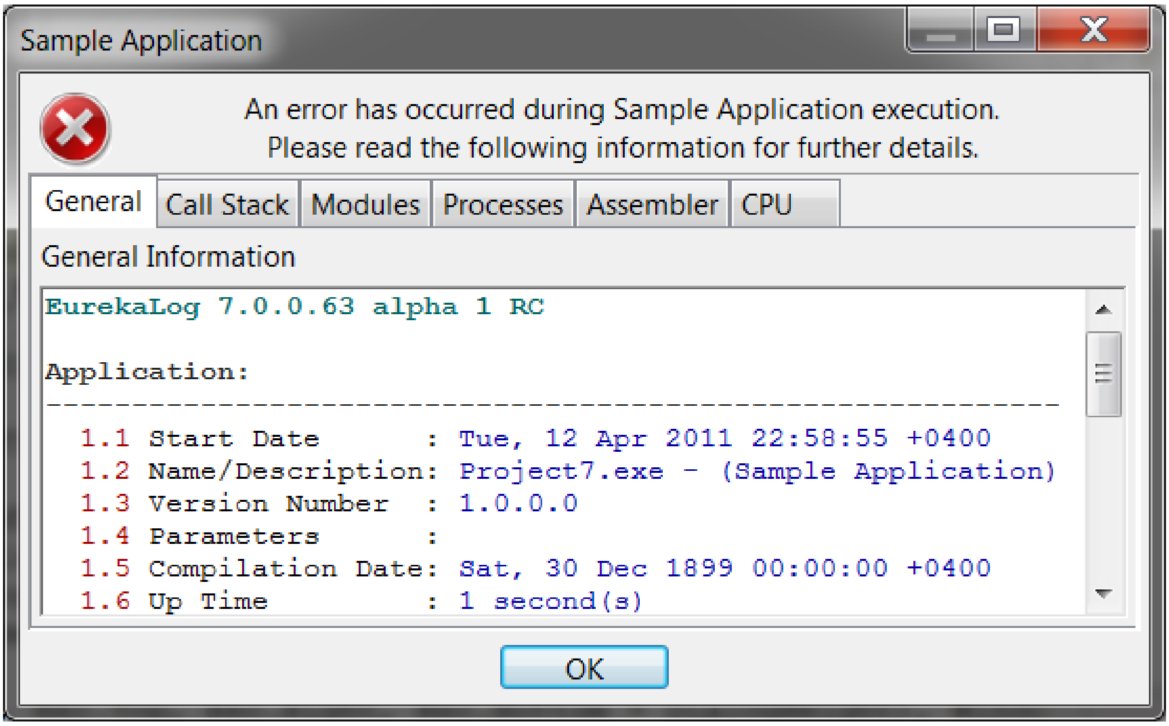
In the example below: the FileDescription field of main .exe contains 'Sample Application' string.



An error has occurred during application execution.
Please read the following information for further details.

```
EurekaLog 7.0.0.63 alpha 1 RC

Application:
----------------------------------------------------------
  1.1 Start Date      : Tue, 12 Apr 2011 22:58:16 +0400
  1.2 Name/Description: Project7.exe - (Sample Application)
  1.3 Version Number  : 1.0.0.0
  1.4 Parameters      :
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0400
  1.6 Up Time         : 2 second(s)
```

"Replace 'application' with real application name" option is unchecked



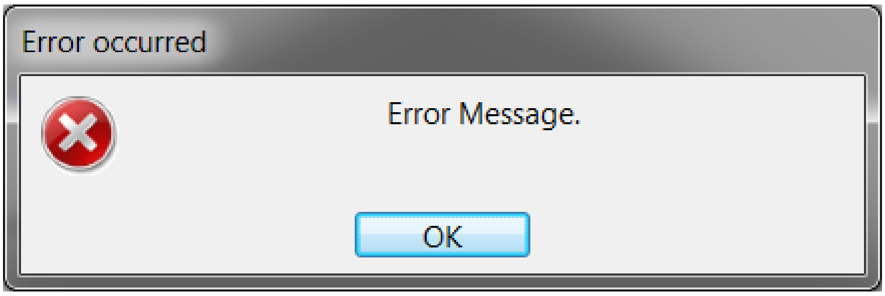
An error has occurred during Sample Application execution.
Please read the following information for further details.

```
EurekaLog 7.0.0.63 alpha 1 RC

Application:
----------------------------------------------------------
  1.1 Start Date      : Tue, 12 Apr 2011 22:58:55 +0400
  1.2 Name/Description: Project7.exe - (Sample Application)
  1.3 Version Number  : 1.0.0.0
  1.4 Parameters      :
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0400
  1.6 Up Time         : 1 second(s)
```

"Replace 'application' with real application name" option is checked

Usually this option is used together with "Replace error icon with real application icon" option (see below) to get a fully personalized view.

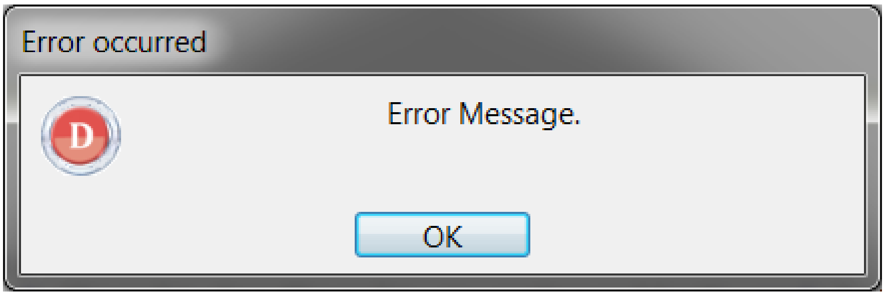
19. "**Replace error icon with real application icon**" (`.edoUseRealIcon`) option will replace standard IDI_ERROR icon to icon of the main .exe (i.e. the first icon). For example:



Error occurred

Error Message.

"Replace error icon with real application icon" option is unchecked



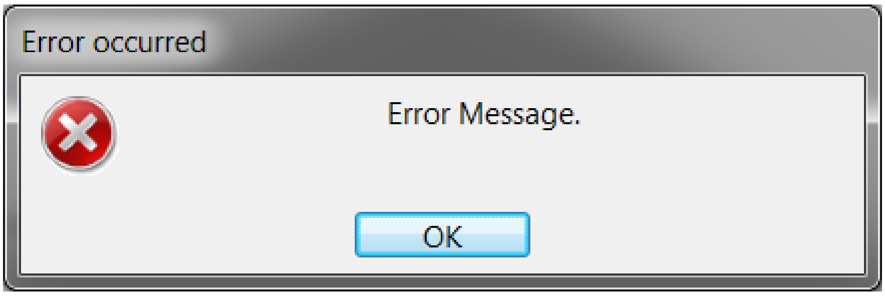
Error occurred

Error Message.

"Replace error icon with real application icon" option is checked

Usually this option is used together with "Replace 'application' with real application name" option (see above) to get a fully personalized view.

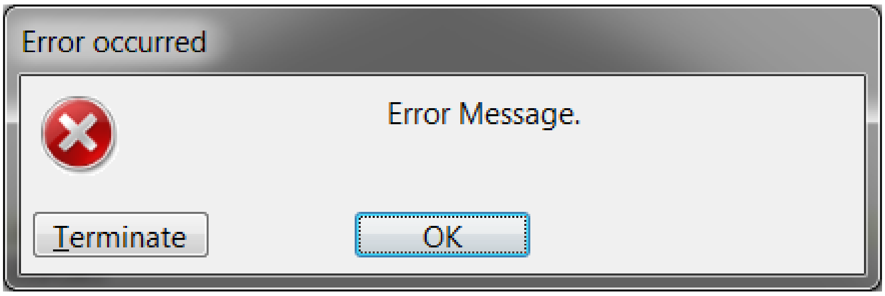
20. **"Do nothing" / "Show 'Restart' checkbox" / "Show 'Terminate' checkbox" after N errors** (`.TerminateBtnOperation`, `.ErrorsNumberToShowTerminateBtn`, `.edoRestartChecked`) options controls the visibility of "restart" and "terminate" checkboxes:



Error occurred

Error Message.

"Do nothing" option is selected or error count is less than the specified count.



Error occurred

Error Message.

"Show 'Terminate' checkbox" options is selected and error count is more than the specified count.

The "Checked" option controls if this checkbox is initially checked or unchecked. If "Checked" option is checked, then when EurekaLog dialog appears with restart/terminate button visible - this button will be default. And the opposite: if the "Checked" option is not checked, then a default button will be OK button, when dialog appears. Of course, this option have no effect, if button is not visible (for example, if you select "Do nothing" option).

A "after N errors" part is controls, when to show restart/terminate checkbox. Again, this part is ignored, if you select "Do nothing" option. Value of 0 means that restart/terminate button will be visible always. Value of 1 means that button will be displayed only in second error dialog. Value of 2 means that you got two error dialog without button and 3rd dialog will have it. And so on.

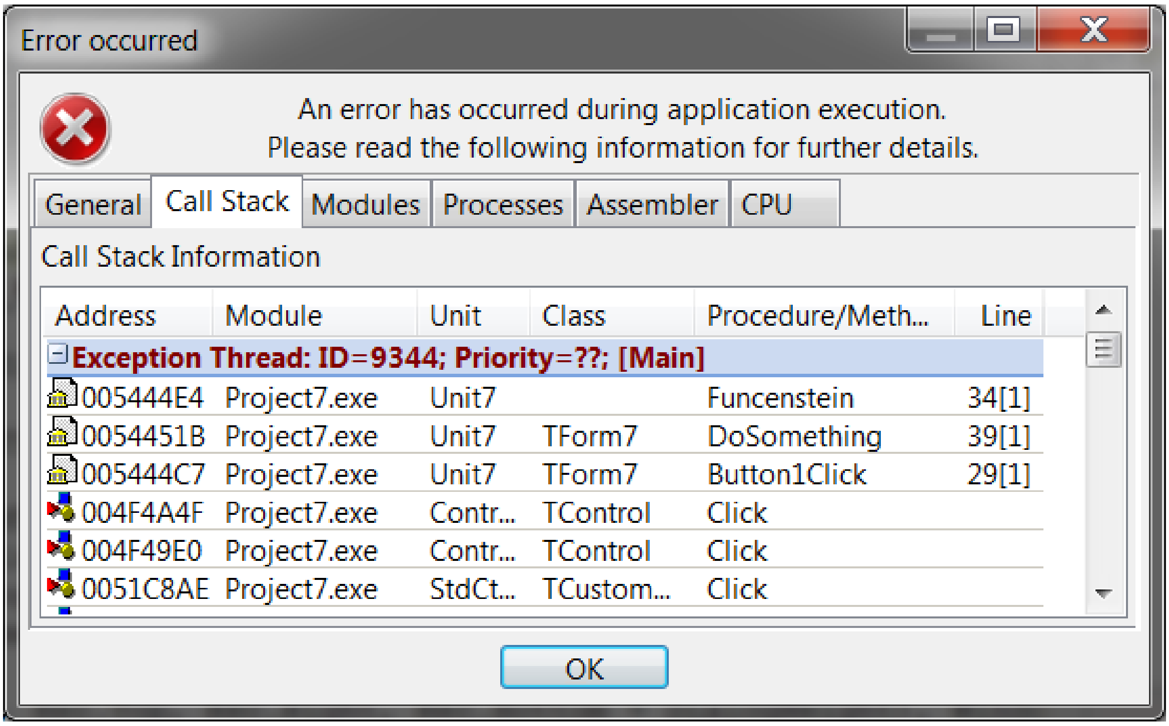
At run-time: when user clicks restart/terminate button, then your application will be restarted or terminated after closing error dialog and saving and sending bug report (if enabled). If user clicks OK button - then your application will continue to run.

21. "**Close every exception dialog after M seconds**" (`.AutoCloseDialogSecs`) option allows you to automatically close exception dialogs after timeout of inactivity. Value of 0 means disabling of such feature - i.e. each dialog may be closed only manually by user (via clicking on dialog's buttons). Any other value (> 0) means that dialog will be closed after that amount of seconds, passed since its popup. For example, if you specify 180 seconds - then dialog will be closed exactly after 3 minutes, if user didn't closed it before.

This option is useful to auto-close error dialogs in possible non-interactive scenarios. Note, that it may be preferable to select other dialog type (non-GUI) in such cases.

Note: this option specifies the delay of user's **inactivity**. If user moves mouse, presses buttons or generates any other input in dialog - auto-close timer will reset. Therefore, setting this option to, say, 5 seconds does not mean that dialog will be closed after 5 seconds. It may stay for, say, 1 minute - but only if user is working with it. Dialog will be closed after 5 seconds if user is away (not moving mouse, etc.).

22. "**Foreground tab**" (`.ForegroundTab`) specifies the default active tab in detailed view when dialog is showed. For example:



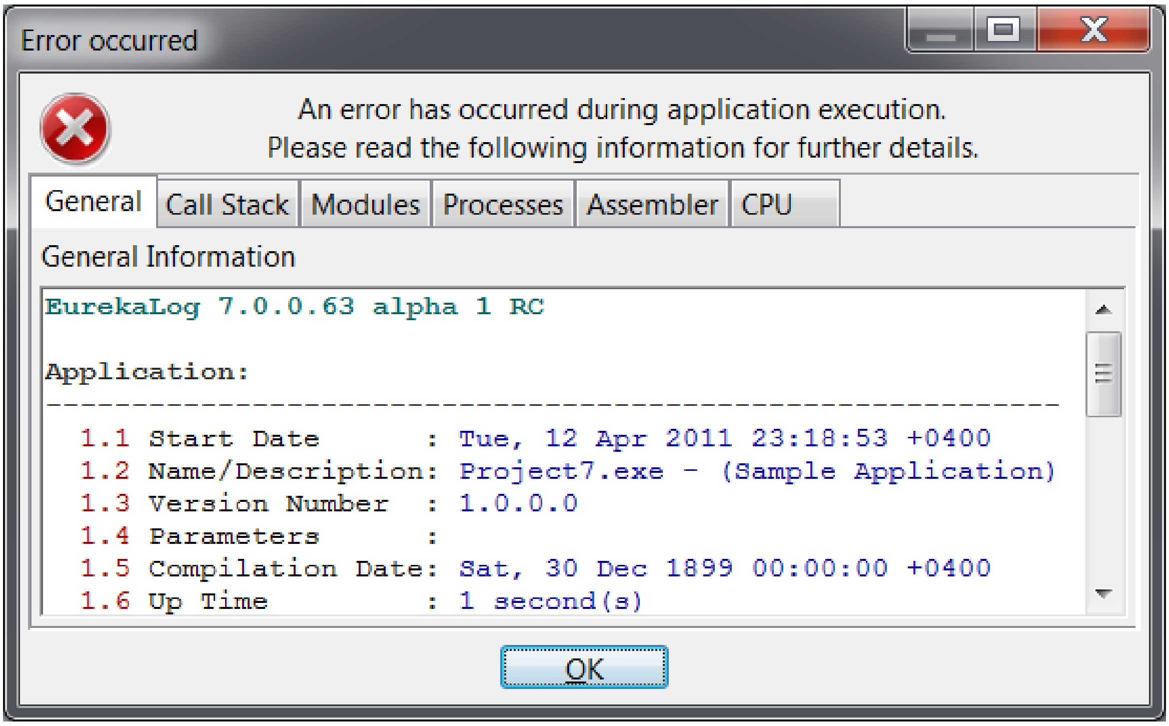
Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
Exception Thread: ID=9344; Priority=??; [Main]					
005444E4	Project7.exe	Unit7		Funcenstein	34[1]
0054451B	Project7.exe	Unit7	TForm7	DoSomething	39[1]
005444C7	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F4A4F	Project7.exe	Contr...	TControl	Click	
004F49E0	Project7.exe	Contr...	TControl	Click	
0051C8AE	Project7.exe	StdCt...	TCustom...	Click	

Foreground tab is set to "Call-stack"



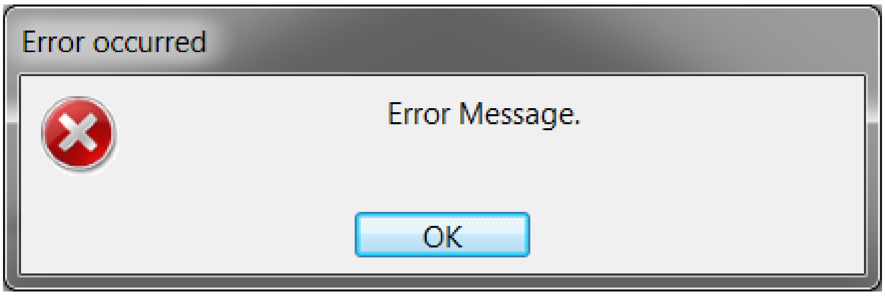
Error occurred
An error has occurred during application execution.
Please read the following information for further details.
General
Call Stack
Modules
Processes
Assembler
CPU
General Information
OK

```
EurekaLog 7.0.0.63 alpha 1 RC

Application:
-----------------------------------------------------------
  1.1 Start Date      : Tue, 12 Apr 2011 23:18:53 +0400
  1.2 Name/Description: Project7.exe - (Sample Application)
  1.3 Version Number  : 1.0.0.0
  1.4 Parameters      :
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0400
  1.6 Up Time         : 1 second(s)
```

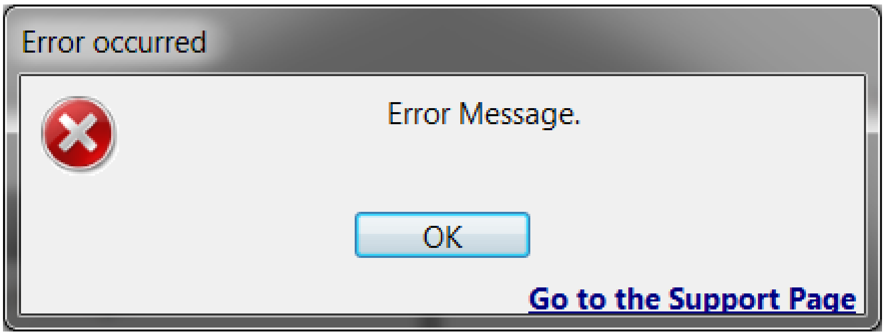
Foreground tab is set to "General"

This option has no effect if detailed mode is never shown.

23. "**Support URL**" (.SupportURL) option allows you to specify an URL, which will be shown in error dialog as hyper-link. Empty value means "do not show hyper-link". For example:



"Support URL" option is empty



"Support URL" option is set to `http://www.example.com/`

At run-time: user can click on that hyper-link and EurekaLog will open a default browser with specified URL. Dialog itself will not be closed.

Note: support URL may include any variable 413. For example, you may specify the following

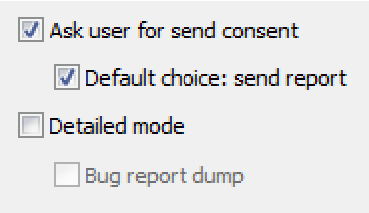
URL: "`https://bugs.example.com/view.php?find=%_BugID%`" (without quotes). When user click on support URL, EurekaLog will open the following URL in user's default browser: `https://bugs.example.com/view.php?find=CFAC0000` (where "CFAC0000" - is just an example of Bug ID 421; it will be different in your case). This allows you to dynamically customize support page.

See also:

- EurekaLog dialog 379 for general description of this dialog's type
- Nested (chained) exceptions 244

10.3.4.4 Console

This is setup options for console dialog 382 (`edtConsole`, `edtConsoleDetailed`, `edtConsoleDump`). They are located at Dialogs tab 267.



☑ Ask user for send consent

☑ Default choice: send report

□ Detailed mode

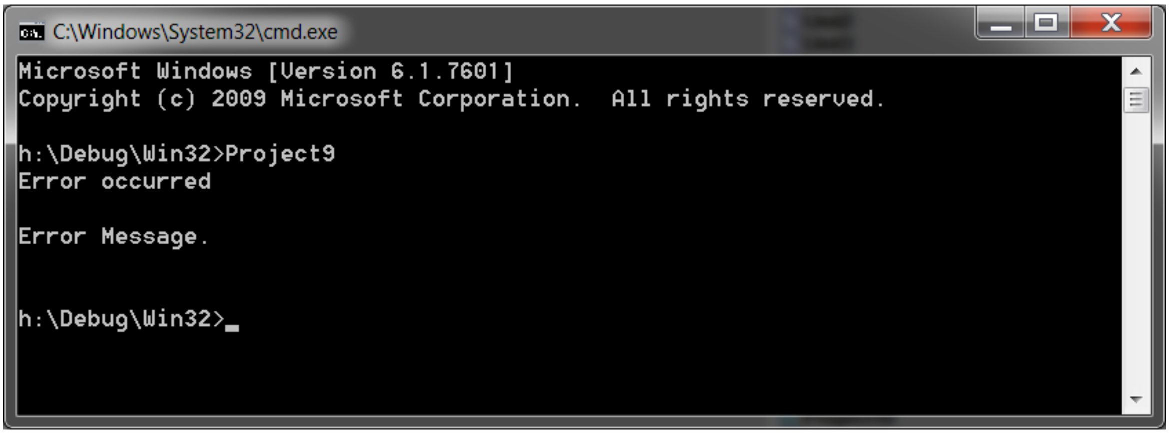
□ Bug report dump

Console dialog options

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

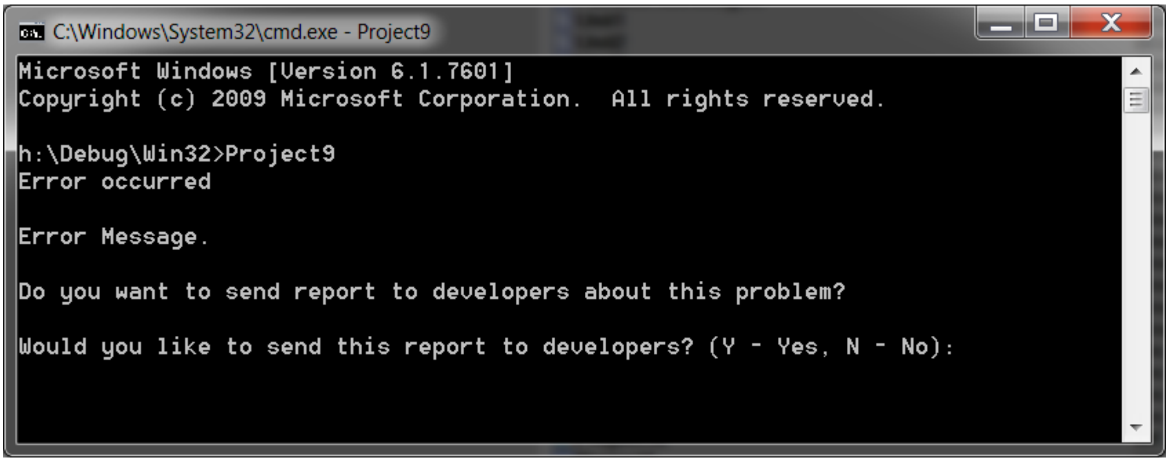
1. "**Ask user for send consent**" (`.edoShowSendErrorReportOption`) option will ask user for their consent before sending bug report to developer - by showing "Do you want to send report to developers about this problem?" message.

This option has no effect if you haven't specified any sending methods. In this case you'll see only error message. For example:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.


h:\Debug\Win32>_
```

Asking for consent is unchecked or there is no sending method available



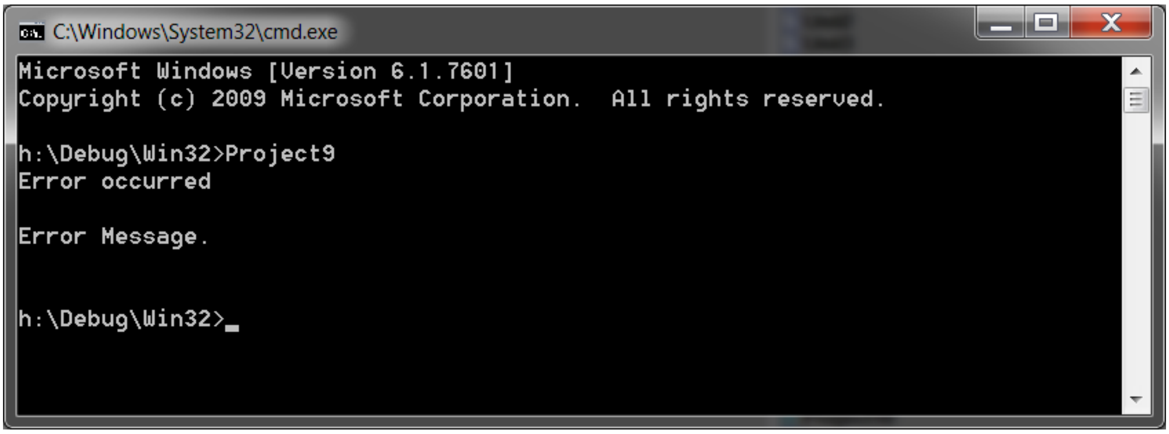
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

Do you want to send report to developers about this problem?

Would you like to send this report to developers? (Y - Yes, N - No):
```

Asking for consent is checked and sending method present

This option will be automatically disabled, if you use output redirection at run-time.

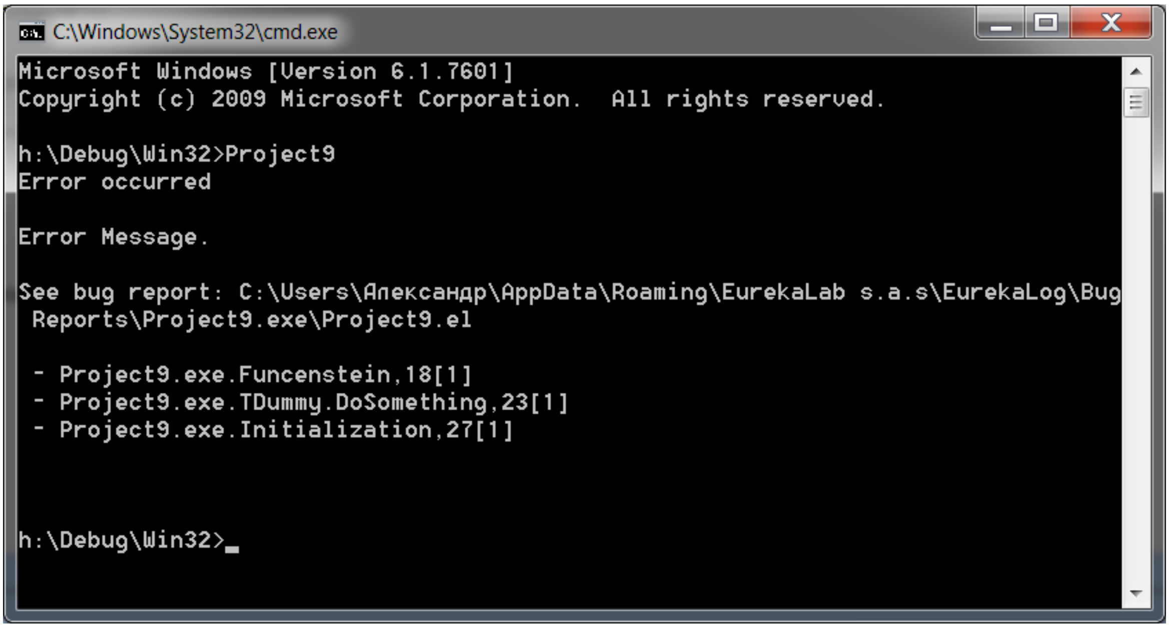
2. Default choice is selected by enabling/disabling "**Default choice: send report**" (`.edoSendErrorReportChecked`) option. If this option is checked - the default choice is to send the report. If this option is unchecked - then the default choice is NOT to send the report.

3. "**Detailed mode**" (`.ExceptionDialogType = edtConsoleDetailed`) option includes a compact call stack to the error message. For example:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.


h:\Debug\Win32>_
```

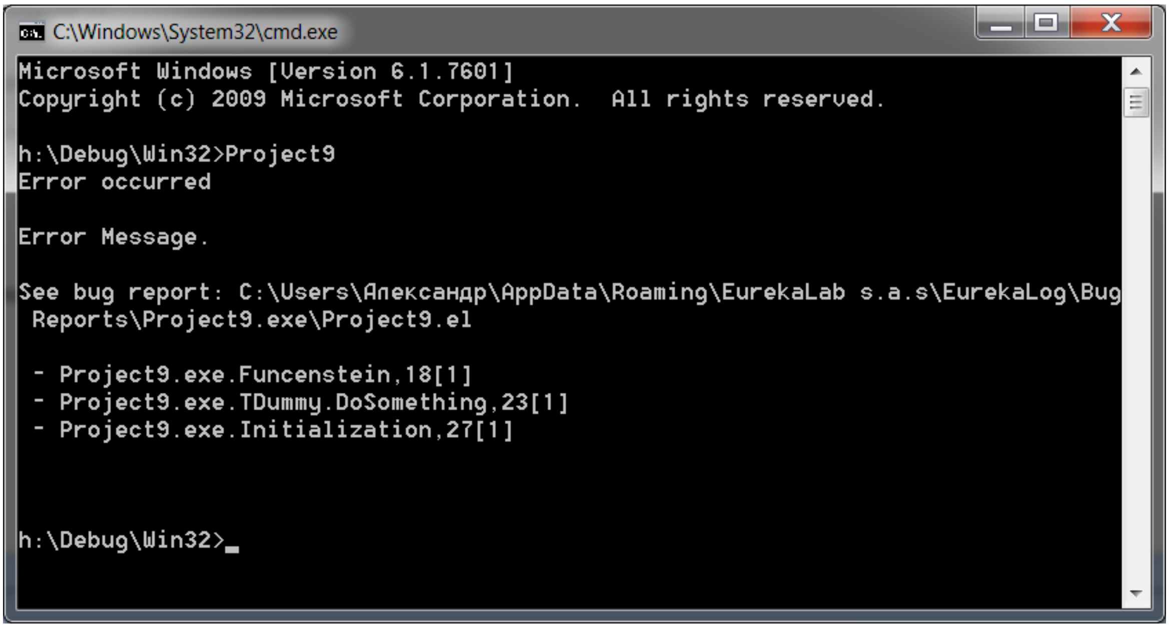
Standard mode



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

See bug report: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Bug
 Reports\Project9.exe\Project9.el

 - Project9.exe.Funcenstein,18[1]
 - Project9.exe.TDummy.DoSomething,23[1]
 - Project9.exe.Initialization,27[1]



h:\Debug\Win32>_
```

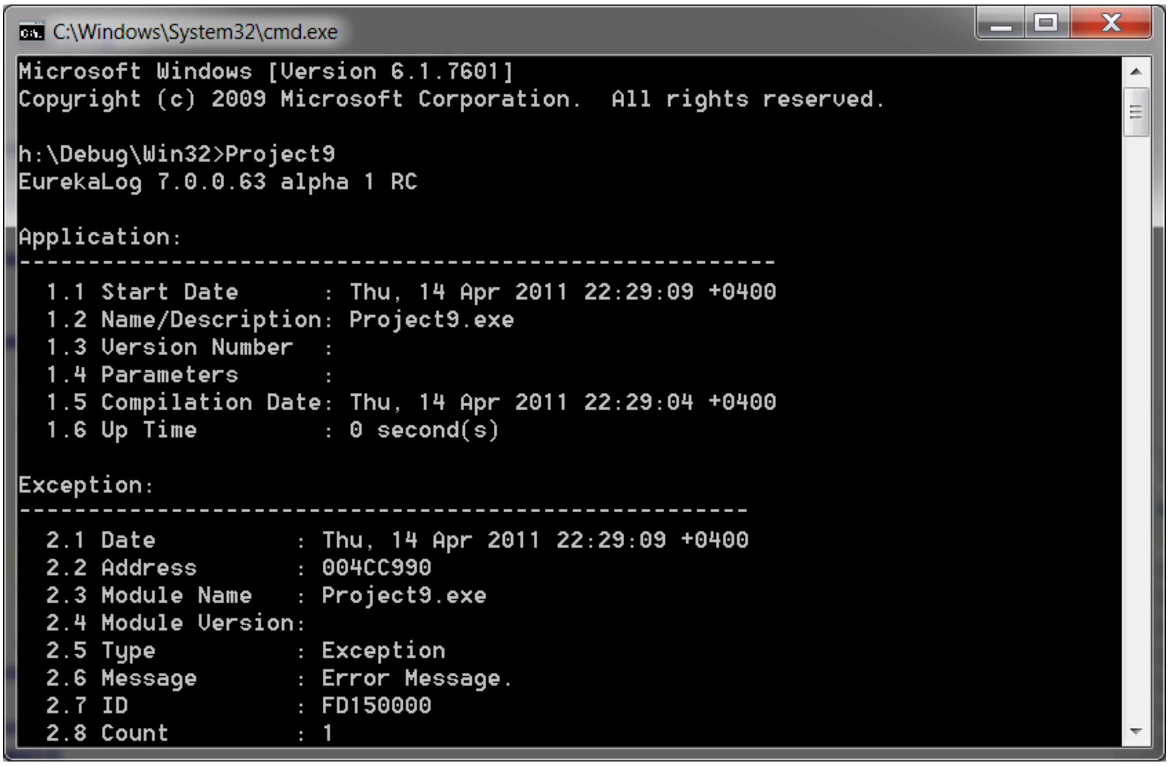
Detailed mode

4. "**Bug report dump**" (`.ExceptionDialogType = edtConsoleDump`) option replaces error message and a call stack with a copy of the bug report. For example:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

See bug report: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Bug
 Reports\Project9.exe\Project9.el

 - Project9.exe.Funcenstein,18[1]
 - Project9.exe.TDummy.DoSomething,23[1]
 - Project9.exe.Initialization,27[1]



h:\Debug\Win32>_
```

Dump mode is unchecked



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
EurekaLog 7.0.0.63 alpha 1 RC

Application:
-------------------------------------------------------
  1.1 Start Date      : Thu, 14 Apr 2011 22:29:09 +0400
  1.2 Name/Description: Project9.exe
  1.3 Version Number  :
  1.4 Parameters      :
  1.5 Compilation Date: Thu, 14 Apr 2011 22:29:04 +0400
  1.6 Up Time         : 0 second(s)

Exception:
-----------------------------------------------------
  2.1 Date          : Thu, 14 Apr 2011 22:29:09 +0400
  2.2 Address       : 004CC990
  2.3 Module Name   : Project9.exe
  2.4 Module Version:
  2.5 Type          : Exception
  2.6 Message       : Error Message.
  2.7 ID            : FD150000
  2.8 Count         : 1
```

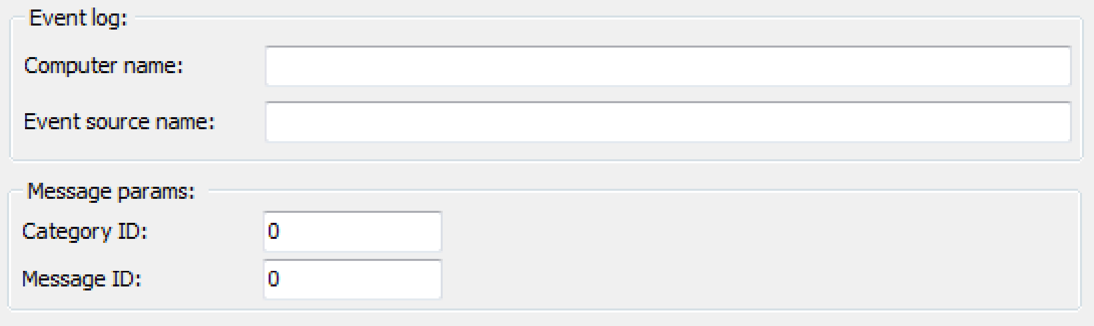
Dump mode is checked

If you enable this mode - it may be a good idea to disable some parts of bug report (especially modules and processes lists), because large bug report will not fit into console. This may not matter, if you use output redirection.

See also:
- Console dialog 382 for general description of this dialog's type
- Nested (chained) exceptions 244

10.3.4.5 System log reporting

This is setup options for system log dialog 384 (edtService). They are located at Dialogs tab 267.



Event log:

Computer name:

Event source name:

Message params:

Category ID: 0

Message ID: 0

System log dialog options

Warning: you must register your application as event source 537 before using this dialog.

1. "**Computer name**" (.EventLogComputer) option identifies a computer to log error message. This is the first parameter to be passed to OpenEventLog. Leave this option empty to use the current machine.

2. "**Event source name**" (.EventLogName) option identifies event source to log error

message. This is the second parameter to be passed to OpenEventLog. This is the name which you've used during your application's registration 537 in system event log. This parameter is required.

3. "**Category ID**" (`.EventLogCategory`) option is optional parameter which specifies a category for error message. Leave it empty if you don't use categories. Category ID is established during application's registration 537.

4. "**Message ID**" (`.EventLogEventID`) option identifies a message to use when logging error message. This parameter is required. Message ID is established during application's registration 537.

To use system event log you must do at least the following:

- Register your application in system event log 537.
- Specify "event source name" in system log reporting dialog.
- Specify "message ID" in system log reporting dialog 543.

This is absolute minimum to make it to work.

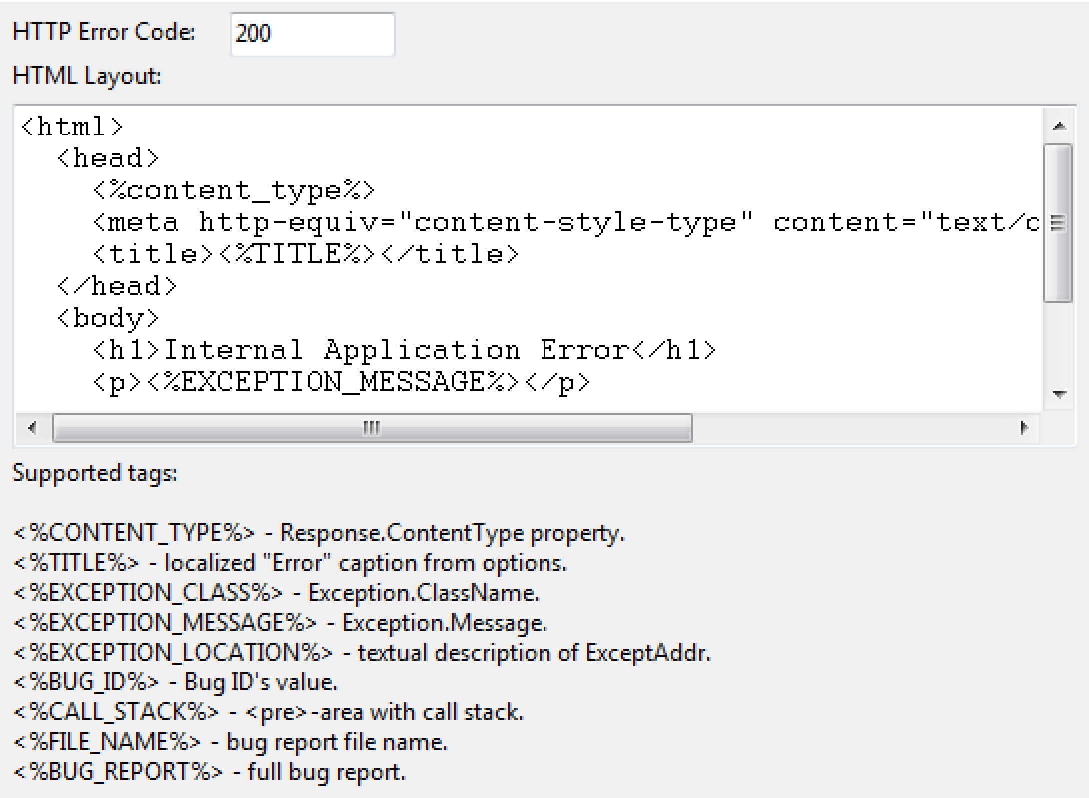
Tip: Message ID usually have form of `$C002`*XXXX*. Where *XXXX* is ID of your event (such as 1, 2, etc.) in hexadecimal form.

See also:

- Configuring system log dialog 543
- System log dialog 384 for general description of this dialog's type
- Setup system logging 535
- Registering event source 537

10.3.4.6 WEB

This is setup options for WEB dialog 386 (`edtWEB`). They are located at Dialogs tab 267.



HTTP Error Code: 200

HTML Layout:

```
<html>
  <head>
    <%content_type%>
    <meta http-equiv="content-style-type" content="text/c
    <title><%TITLE%></title>
  </head>
  <body>
    <h1>Internal Application Error</h1>
    <p><%EXCEPTION_MESSAGE%></p>
```

Supported tags:

<%CONTENT_TYPE%> - Response.ContentType property.
<%TITLE%> - localized "Error" caption from options.
<%EXCEPTION_CLASS%> - Exception.ClassName.
<%EXCEPTION_MESSAGE%> - Exception.Message.
<%EXCEPTION_LOCATION%> - textual description of ExceptAddr.
<%BUG_ID%> - Bug ID's value.
<%CALL_STACK%> - <pre>-area with call stack.
<%FILE_NAME%> - bug report file name.
<%BUG_REPORT%> - full bug report.

WEB dialog options

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

1. "**HTTP Error Code**" (.WebErrorCode) option specifies value of HTTP Status Code to be returned to client's web browser when exception is occurred.

Most typical values are either 200 (default) or 500:
- 200 status code means success/OK, and it is a usual status code to be returned to client when web application produces content page without any error. Use this status code to produce error message as normal web pages.
- 500 error code means internal server error, and it is a typical error code to be returned when something goes wrong with web application. Use this or any other error code to indicate failure to client's web browser.

Important note: some web application implementations (server or browser) may ignore actual page content for status codes like 500 error code. This means that customized HTML page (see "**HTML layout**" option below) will be ignored. Use 200 status code for such cases.

Note: not all web application supports returning custom status code for error page. Support for this feature depends on used framework, its version and its configuration.

2. "**HTML layout**" (.HTMLLayout) option specifies an HTML page template to be send to client in case of error during its request's processing. E.g. this is an error HTML page. You can put any text here, the resulting page will be exactly the same as you specified here. You can customize error HTML page to match your web-site.

The target text's encoding is determinated by meta "content-type" HTML tag (note "charset=UTF-8" in this example):

```
<head>
   <meta http-equiv="content-type" content="TEXT/HTML;charset=UTF-8" />
</head>
```

It can be anything supported by the host OS. For example: iso-8859-1, Windows-1252, windows-1251 or even UTF-8, unicode or unicodeFFFE. See this list of supported encodings for Windows platform.

Notes:
- It's a good idea to use either <%CONTENT_TYPE%> (see below) or fixed "content-type" HTML meta tag with UTF-8 encoding.
- If you specify fixed encoding - then dialog will adjust HTTP headers as needed.
- Old Delphi versions supports only ANSI encodings. So even if you specify one of unicode encodings (like UTF-8 or any other) - you will be able to show only characters in current code page (even though result page will be properly encoded in unicode). You won't be able to use full range of unicode and mix, say, Latin, Japanese and Cyrillic in the same text.

If encoding is missing in HTML page template - then dialog will use encoding from Response object. If it's missed too - then dialog will use UTF-8 as default.

Page customization via tags

You can also use some special tags, which looks like this: *<%TAG%>*. If you insert such tag in template's text, it will be replaced with actual value at run-time. There is a hint for common used tag right at dialog's settings page.

Note: tags are case-insensitive.

Supported tags:

<%CONTENT_TYPE%>
This tag is replaced with Response.ContentType property. Use this tag in <head> part to indicate proper encoding. You can enter fixed meta "content-type" HTML tag or you can use

<%CONTENT_TYPE%> to indicate current encoding.

Example:

```
      <meta http-equiv="content-type" content="TEXT/HTML;charset=UTF-8" />
```

Note that <%CONTENT_TYPE%> is expanded to full meta content HTML tag, not just to "charset=X" part.

EurekaLog 7 Documentation
It's standard error caption for error dialogs. It can be customized on localization page 339. Usually it's used in <title> HTML tag, but it actually can be used anywhere.

Example:

Error occurred

<%EXCEPTION_CLASS%>
It's class name of exception object. It can be used anywhere.

Example:

EAccessViolation

<%EXCEPTION_MESSAGE%>
It's exception message. It can be used anywhere.

Example:

Access violation at address 0216942E in module 'ISAPI.dll'. Write of address 00000000

<%EXCEPTION_LOCATION%>
This is a short textual description for exception address. Indicates point of exception's raising. It can be used anywhere.

Example:

(000D842E){ISAPI.dll } [0216942E] MainISAPI.Error (Line 41, "MainISAPI.pas") + $2

<%BUG_ID%>
This is Bug ID 421 value from bug report. It's extremely useful to identify a problem in bug report file. Can be used anywhere.

Example:

824B0000

We recommend that you use the following model in the production for security reasons: don't expose any information on error page, place only a generic message like:

```
<p>The server application has encountered an error with <b><%BUG_ID%> code</b>.
<p>Please, <a href="mailto:your-account@example.com">contact server's administra
```

You can analyze full bug report from logs (bug report file) - as administrator.

Note: even if you don't specify this tag anywhere - dialog will automatically append a hidden comment to page's source with Bug ID's value.

For example:

```
    ... (other page content)
  </body>
</html>
<!-- EurekaLog page ID: CC2F96D8 -->
```

```
<!-- EurekaLog Bug ID: 824B0000 -->
```

Page ID is just a random number to distinct one page from another. Bug ID is Bug ID's value.

<%CALL_STACK%>
This is a compact form of call stack from bug report. Can be useful for quick diagnostic. **It's highly recommended to hide this information in release version of your application for security reasons.** The call stack is wrapped in <pre> HTML tag, so it can be used only inside <body> HTML tag.

Example:

```
<pre>
 - ISAPI.dll.MainISAPI.Error,41[4]
 - ISAPI.dll.MainISAPI.GoToError,82[1]
 - ISAPI.dll.MainISAPI.RaiseException,87[1]
 - ISAPI.dll.MainISAPI.TModule.Action,97[5]
 - ISAPI.dll.HTTPApp.TWebActionItem.DispatchAction
 - ISAPI.dll.HTTPApp.TCustomWebDispatcher.DispatchAction
 - ISAPI.dll.HTTPApp.TCustomWebDispatcher.HandleRequest
 - ISAPI.dll.HTTPApp.TDefaultWebAppServices.InvokeDispatcher
 - ISAPI.dll.HTTPApp.TDefaultWebAppServices.HandleRequest
 - ISAPI.dll.WebReq.TWebRequestHandler.HandleRequest
 - ISAPI.dll.ISAPIApp.TISAPIApplication.HttpExtensionProc
 - ISAPI.dll.ISAPIApp.HttpExtensionProc
</pre>
```

<%FILE_NAME%>
Full file name to bug report file. Can be used anywhere.

Example:

C:\inetpub\wwwroot\logs\ISAPI.el

<%BUG_REPORT%>
Full bug report enclosed in <pre> HTML tag.

WARNING: never show this information in production for security reasons.

Example:

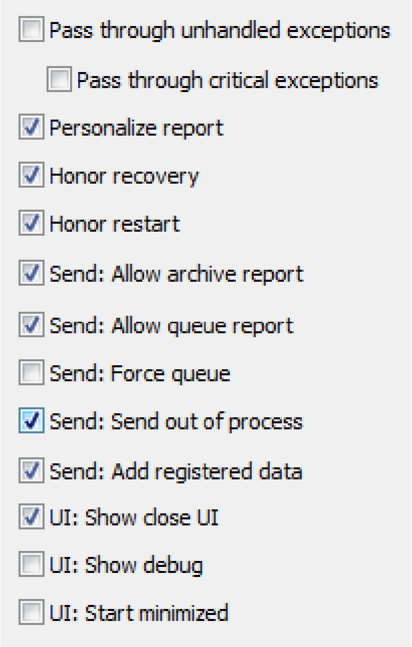
```
<pre>EurekaLog 7.0.0.63 alpha 1 RC

Application:
-------------------------------------------------------
  1.1 Start Date      : Fri, 15 Apr 2011 22:34:24 +0359
  1.2 Name/Description: w3wp.exe - (IIS Worker Process)
  1.3 Version Number  : 7.5.7601.17514
  1.4 Parameters      : -m 0 -t 20
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0359
  1.6 Up Time         : 10 minute(s), 7 second(s)

Exception:
--------------------------------------------------------------------------------
  2.1 Date          : Fri, 15 Apr 2011 22:44:32 +0359
  2.2 Address       : 0236942E
  2.3 Module Name   : ISAPI.dll
  2.4 Module Version:
  2.5 Type          : EAccessViolation
  2.6 Message        : Access violation at address 0236942E in module 'ISAPI.dll'
  2.7 ID             : 7D390000
  2.8 Count          : 1
  2.9 Status         : New
  2.10 Note          :

... (other information in bug report - cut for compactness)

00D3EFF4: 00000000   023694FE: FF C3 E9 FF CA F2 FF EB F0 8D 45 E8 50 A1 70 DB
00D3EFF8: 00000000   0236950E: 37 02 89 45 E0 C6 45 E4 00 8D 55 E0 33 C9 B8 DC
00D3EFFC: 00D3F018   0236951E: 95 36 02 E8 92 43 F5 FF 8B 45 E8 B9 4C 00 00 00
</pre>
```

<%HTML_TAG%>
This is obsolete tag used only for backward compatibility. Do not use it in new applications.

See also:
- WEB dialog 386 for general description of this dialog's type
- Nested (chained) exceptions 244

10.3.4.7 Windows Error Reporting

This is setup options for WER dialog 389 (edtWER). They are located at Dialogs tab 267.



☐ Pass through unhandled exceptions
  ☐ Pass through critical exceptions
☑ Personalize report
☑ Honor recovery
☑ Honor restart
☑ Send: Allow archive report
☑ Send: Allow queue report
☐ Send: Force queue
☑ Send: Send out of process
☑ Send: Add registered data
☑ UI: Show close UI
☐ UI: Show debug
☐ UI: Start minimized

WER dialog options

1. "**Pass through unhandled exceptions**" (.WERPassThroughUnhandled) option enables special mode for real unhandled exceptions 40. Once checked, this option will redirect such exceptions directly to the system, ignoring any application's processing code. All other exceptions will be processed as usual.

In other words, if you check this option - then all your customizations will be ignored (for real unhandled exceptions only).

This option is useful to get exact state of the application by using minidumps 573, since all processing are ignored. So you can get the closest possible image of the problem.

This option requires low-level hooks injection in order to work properly.

2. "**Pass through critical exceptions**" (`.WERPassThroughUnexpected`) option do the same thing as "Pass through unhandled exceptions", but do this for "critical" exceptions. Critical exception are the exceptions which you should not handle at all. This includes exceptions such as access violations. All other exceptions (like EStreamError) will behave as usual.

Warning: you won't be able to catch critical exceptions with try/except blocks. Bug report will be send and application will be terminated immediately upon critical exception raising. All other exceptions will behave normally.

This option has no effect if "Pass through unhandled exceptions" option is not checked.

3. "**Personalize report**" (`.WERCustomizeReport`) option will report basic information and description of the application to the system error processing. If not checked - system will extract information on its own. Usually the results of both methods are very close.

4. "**Honor recovery**" (`.WERSubmitHonorRecovery`) option follows any recovery registration for the application. This option is related to Restart & Recovery API in Windows Vista+. If this option is unchecked - WER will not perform any registered recovery activities.

5. "**Honor restart**" (`.WERSubmitHonorRestart`) option follows any restart registration for the application. This option is related to Restart & Recovery API in Windows Vista+. If this option is unchecked - WER will not perform any registered restart activities.

6. Unchecking the "**Allow archive report**" (`.WERSubmitNoArchive`) option will disable report's archiving.

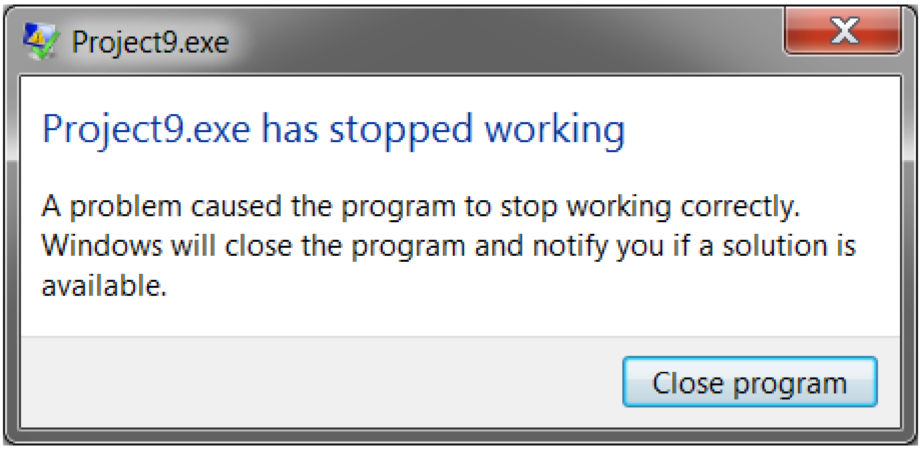
7. "**Allow queue report**" (`.WERSubmitNoQueue`) option allows to queue report for later send, if sending is not possible now. If this option is unchecked - report will never be queued. If there is adequate user consent the report is sent to Microsoft immediately; otherwise, the report is discarded. The report is discarded for any action that would require the report to be queued. For example, if the computer is offline when you submit the report, the report is discarded. Also, if there is insufficient consent (for example, consent was required for the data portion of the report), the report is discarded.

8. "**Force queue**" (`.WERSubmitQueue`) option adds the report to the WER queue without notifying the user. The report is queued only - reporting (sending the report to Microsoft) occurs later based on the user's consent level.

9. "**Send out of process**" (`.WERSubmitOutOfProcess`) option spawns another process to submit the report. The calling thread is blocked until the function returns.

10. "**Add registered data**" (`.WERSubmitAddRegisteredData`) option adds the data registered by WerSetFlags, WerRegisterFile, and WerRegisterMemoryBlock to the report.

11. "**Show close UI**" (`.WERSubmitNoCloseUI`) option displays dialog this "Close application" button:



Project9.exe

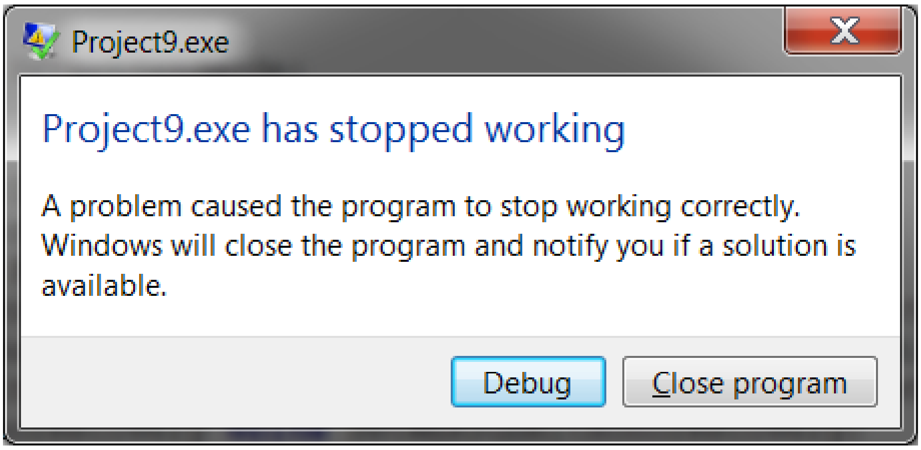
Project9.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

Dialog for the "Show close UI" option

If this option is unchecked - there will be no such dialog. Other dialogs (like sending report and searching for the solution) will be present.

12. "**Show debug**" (`.WERSubmitShowDebug`) option shows a "Debug" button to launch and attach a default debugger.



Project9.exe

Project9.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

"Show debug" option is checked

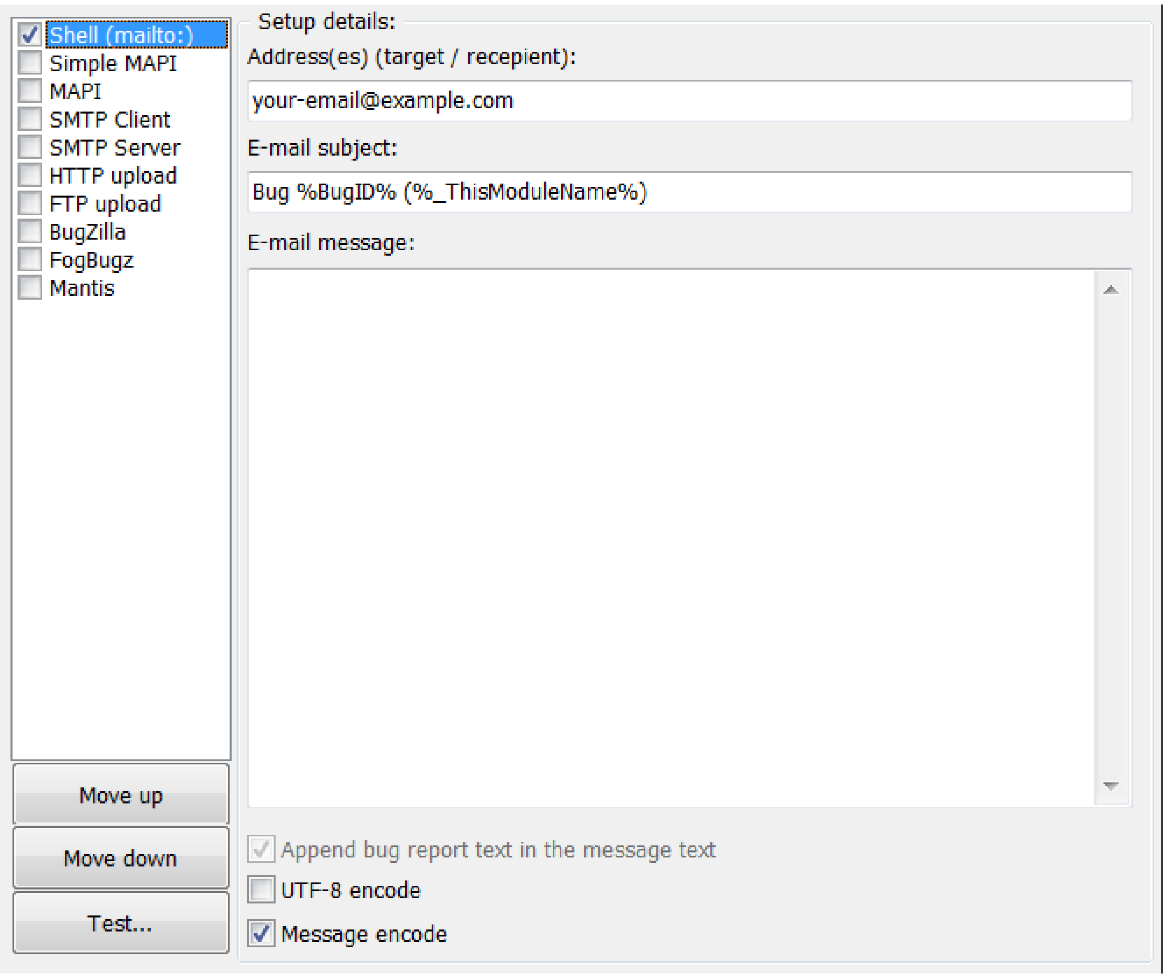
This option has no effect if "Show close UI" option is not checked.

13. "**Start minimized**" (`.WERSubmitStartMinimized`) option runs initial UI as minimized and flashing.

See also:
- WER dialog 389 for general description of this dialog's type
- Using Windows Error Reporting 573
- WERReportSubmit function

10.3.5 Report sending page

This is "Sending" page in EurekaLog project's options 225.



☑ Shell (mailto:)
□ Simple MAPI
□ MAPI
□ SMTP Client
□ SMTP Server
□ HTTP upload
□ FTP upload
□ BugZilla
□ FogBugz
□ Mantis

Setup details:

Address(es) (target / recepient): your-email@example.com

E-mail subject: Bug %BugID% (%_ThisModuleName%)

E-mail message:

☑ Append bug report text in the message text
□ UTF-8 encode
☑ Message encode

Sending tab with "Shell" send engine selected

This page consists of two parts: there is a list of available send engines on the left. The rest of the area is changed depending on your send engine's choice.

You can select (enable) one or several sending methods for your application. If you select more than one method - each send method will be executed one after another until successful send occurs. You can change order of sending via move up/move down buttons.

Select one or more send methods for your application by checking appropriate checkboxes and specify all other options for the specific send method:

- Shell 309
- Simple MAPI 314
- MAPI 315
- SMTP client 316
- SMTP server 319
- HTTP 320
- FTP 321
- FogBugz 322
- Mantis 327
- BugZilla 331
- JIRA 335

You can test sending of currently selected method with current options by clicking on "Test" button in the bottom-left corner.

There are also additional sending options available on sub-category: Sending options 304.

Important Note: Test sending will be carried out by IDE or Setting Editor Tool, not by your real application. This means that IDE/Settings Editor Tool must have Internet access, must not be blocked by firewall. It also means that application and exception names and version will be different from your real application. You can use "Custom/Manual" page 356 to add the following overrides:

```
_BugAppVersion="7.1.1.41"
_BugID="3E860000"
_BugIDSource="settingseditor.exe settingseditor.exe eeurekaconnectiontestexcepti
_BugText="(Error Message) This is a demo bug report from EurekaLog connection te
_BugType="EEurekaConnectionTestException"
_EL_Build="7.1.1.41"
_EL_EMail="example@example.com"
_EL_MachineID="BUILD-PC"
_EL_OSBuild="6002 (6.0.6001.18000)"
_EL_OSType="Microsoft Windows Vista (64 bit)"
_EL_Platform="Windows x86-64"
_EL_StepsToReproduce="(Steps to reproduce) I've clicked on \qTest\q button on \c
```

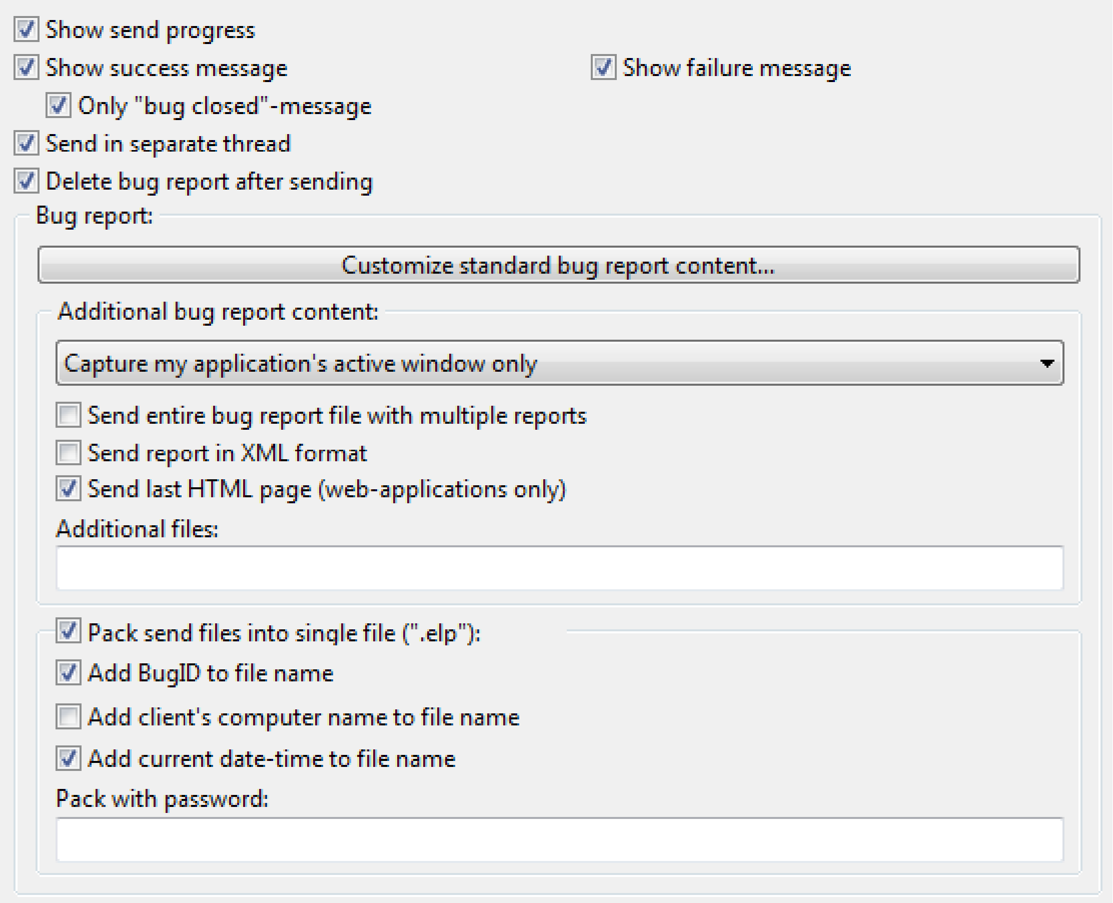
Those overrides will not be saved in your executable, you can use them only for testing.

Note: You may alter send methods at run-time with .SenderClasses property. For example, you may clear .SenderClasses property and then use .AddSenderClass helper method to add one or more send methods.

See also:
- Send engines 390 to get more information about each send method
- Selecting sending method 55
- Security Considerations 158

10.3.5.1 Sending options page

This is "Sending/Options" page in EurekaLog project's options 225.



☑ Show send progress

☑ Show success message

☑ Show failure message

☑ Only "bug closed"-message

☑ Send in separate thread

☑ Delete bug report after sending

Bug report:

Additional bug report content:

Capture my application's active window only

☐ Send entire bug report file with multiple reports

☐ Send report in XML format

☑ Send last HTML page (web-applications only)

Additional files:

☑ Pack send files into single file (".elp"):

☑ Add BugID to file name

☐ Add client's computer name to file name

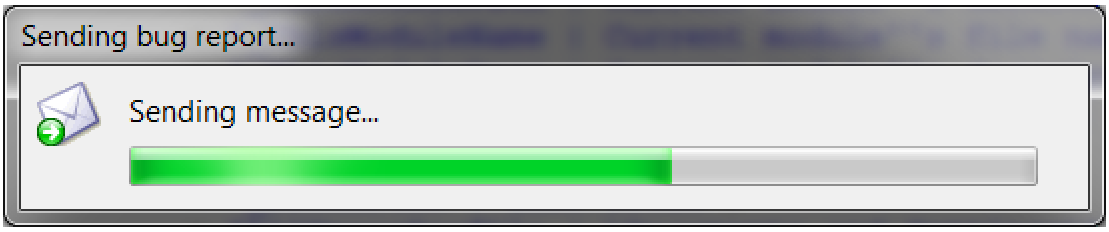
☑ Add current date-time to file name

Pack with password:

Sending options tab

Warning: options on this page have no effect if no send method 302 was selected.

1. "**Show send progress**" (`.sndShowSendDialog`) option enables send progress dialog which indicates sending process:

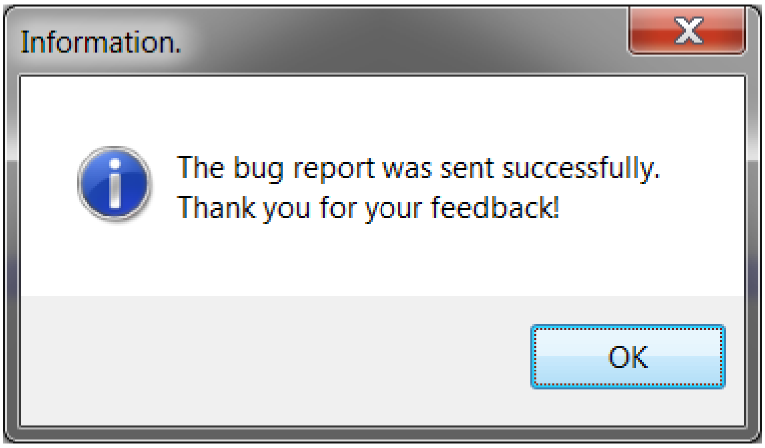


Sending bug report...

Sending message...

You should uncheck this option for non-GUI dialogs.

2. "**Show success message**" (`.sndShowSuccessMsg`) option shows informational message box, if sending was succeeded:

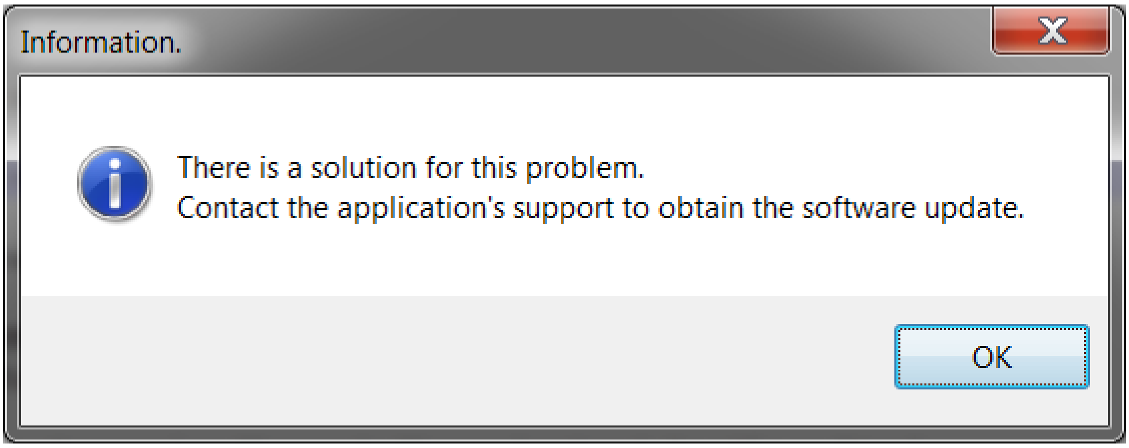

Information.

The bug report was sent successfully.
Thank you for your feedback!

General send success message

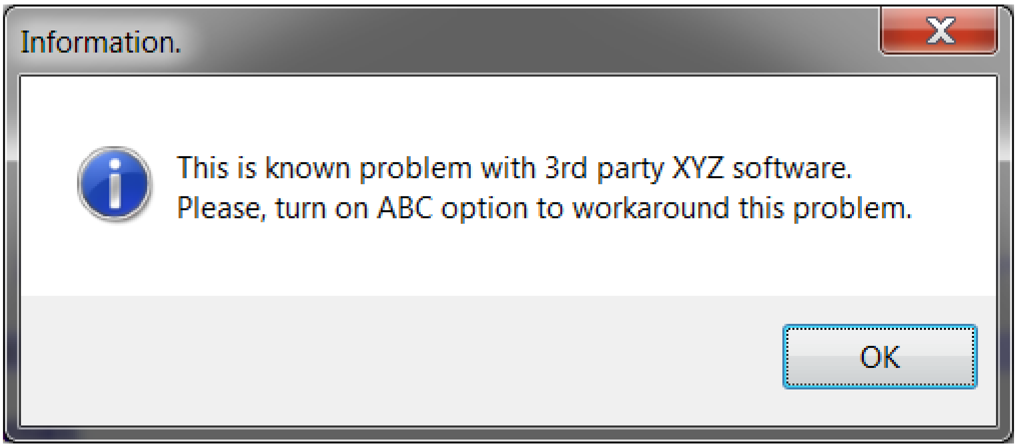
You should uncheck this option for non-GUI dialogs.

3. "**Only "bug closed"-message**" (`.sndShowSuccessBugClosedOnlyMsg`) option alters previous option ("Show success message"). If both "Show success message" and "Only "bug closed"-message" options are checked - EurekaLog will show informational message only if sending was succeeded AND bug in question was closed. The displayed message will show either generic "bug is fixed, update your software" message or custom response message (if supplied):


Information.

There is a solution for this problem.
Contact the application's support to obtain the software update.

General "bug closed" message


Information.

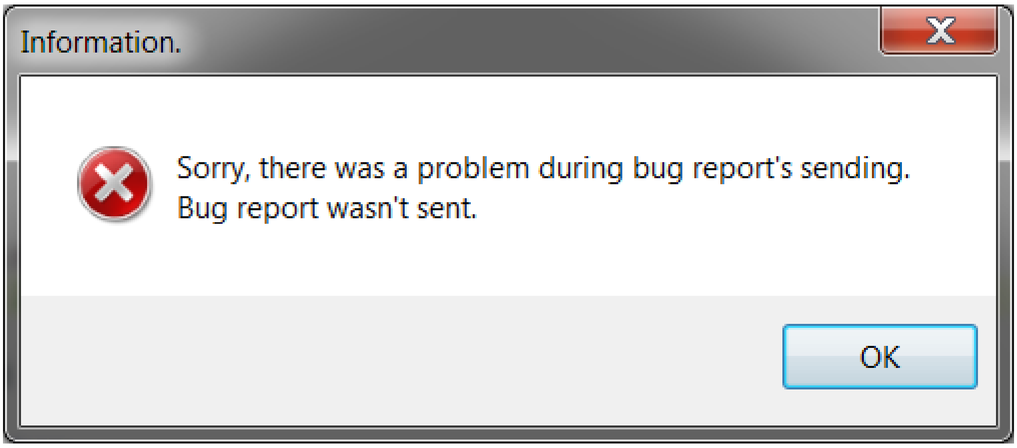
This is known problem with 3rd party XYZ software.
Please, turn on ABC option to workaround this problem.

Custom "bug closed" message
(supplied by bug-tracker)

Notes:

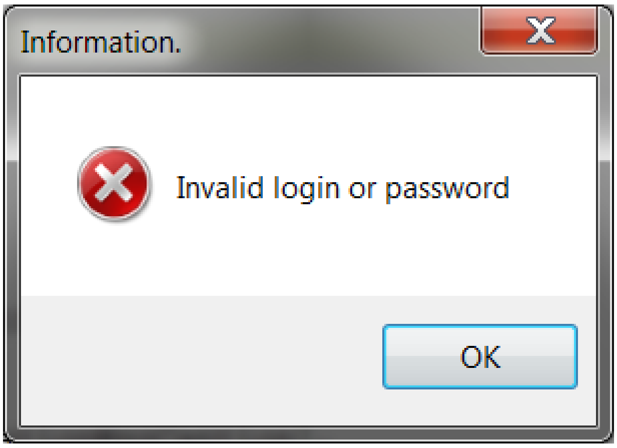
- Not all send methods supports detection of bug report fixing. For example, any e-mail send method is unable to perform such checking.
- Not all send methods supports custom "closed" messages. For example, HTTP upload is unable to do this unless you write a custom script which supplies a custom feedback.
- If "Only "bug closed"-message" option is unchecked - then EurekaLog will show both types of "success" messages.
- If "Show success message" option is unchecked - then "Only "bug closed"-message" option has no effect and EurekaLog will not show any "success" messages.

4. "**Show failure message**" (`.sndShowFailureMsg`) option shows informational message box, if sending has failed:


Information.

Sorry, there was a problem during bug report's sending.
Bug report wasn't sent.

Generic send failure message
(error message is not available)


Information.

Invalid login or password

Send error when error message is available

You should uncheck this option for non-GUI dialogs.

5. "**Send in separate thread**" (`.sndSendInSeparatedThread`) option will send report in background worker thread. Exception thread will wait for this background thread to finish sending. Exception thread will update UI (process messages).

Check this option to get smooth UI during sending.
Uncheck this option is you are not using UI dialogs.

6. "**Delete bug report after sending**" (`.sndDeleteFileAfterSend`) option deletes local bug report file (you can set its saving here 264) after successful sending. If user selected not to send report or if the report sending wasn't successful - bug report file will not be deleted. There is also a similar "Delete file at version change" option 264.

7. "**Screenshot capture**" (.sndScreenshot) option (combobox with different options) specifies how EurekaLog will capture a screenshot:

- "**Do not take screenshot**" (ssNone) - do not capture/store/send screenshot at all. Use this for non-GUI applications or if you don't need screenshot.
- "**Capture active window only (may belong to other process)**" (ssForegroundWindow) - capture window from GetForegroundWindow.
- "**Capture my application's active window only**" (ssActiveWindow) - capture window from GetActiveWindow. Usually it's the best option for typical GUI application.
- "**Capture my application's windows only**" (ssApplication) - capture part of desktop which is bounded to rectangle of all your (visible) windows. It's possible that other applications will be caught in this rectangle. It's recommended option if you have non-modal windows with important information.
- "**Capture primary monitor's workarea**" (ssWorkarea) - capture primary monitor without task bar or any other desktop panels. Some sensitive information from other applications can be caught with this option. Use with care (your application may be classified as spyware).
- "**Capture primary monitor's screen**" (ssPrimary) - capture the entire primary monitor's screen. Some sensitive information from other applications can be caught with this option. Use with care (your application may be classified as spyware).
- "**Capture the entire screen (full desktop)**" (ssDesktop) - capture the entire Windows desktop (multiple monitors). This option works exactly as "Capture primary monitor's screen" on any single-monitor system. Some sensitive information from other applications can be caught with this option. Use with care (your application may be classified as spyware).

Screenshot is sent as separate file in 256-color (8-bit) PNG format (if screenshot creation was enabled, of course). Maximum screenshot file size is typically less than 150 Kb for full screen. Typical file size is around 20 Kb (when saving one average window only).

Notes:

- Active control or active window are indicated by bounding red rectangle.
- Screenshot will contain mouse cursor, if mouse cursor was positioned inside captured screen area.
- Your application's windows may be covered by other applications. This is especially true for "Capture active window only (may belong to other process)" mode.
- Screenshot may contain data from multiple monitors (capturing entire desktop or capturing window which is placed across few monitors). Any area outside of any monitors (if it exists) will be filled with black color.

8. "**Send entire bug report file with multiple reports**" (.sndSendEntireLog) option specifies content of .el/.elx file. If this option is unchecked - then sent bug report file will contain only 1 bug report - the one from current problem. If this option is checked - then bug report file will contain exact copy of locally saved bug report file which may contain some old reports.

This option has no effect, if you do not save bug report locally or if you limit it to 1 bug report.

Note: it's best to enable "Delete bug report after sending" option, if you enabled this option.

9. "**Send report in XML format**" (.sndSendXMLLogCopy) option instructs EurekaLog to send .elx file instead of .el file. .el file is plain-text report. .elx is XML bug report.

10. "**Send last HTML page**" (.sndSendLastHTMLPage) option includes a last HTML page (if available). This option has effect only for web applications (ISAPI, (Win)CGI, IntraWeb, etc).

11. "**Additional files**" (.AttachedFiles) option specifies additional files to include into sent bug report. You can use ';' to separate files. You can use environment variables 413 to access special folders. You can use relative file paths to access application's folder (useful for portable applications). For example: 'Configs\Master.ini;%APPDATA%\MySoftware\Config.ini' - this value will attach two files. One file is Master.ini file in Configs subfolder of exe's directory. Second file will be Config.ini file from roaming application folder.

You can also specify this option at run-time. Or you can use OnZippedFilesRequest event handlers.

Please note that this option will include files *inside* bug report when "**Pack send files into single file**" option is checked. To attach more files near bug report - use OnAttachedFilesRequest event.

Note: current application directory have no effect on relative paths in this option. All relative paths are always resolved with application folder.

12. "**Pack send files into single file**" (.sndPack) option sends one single .elp file instead of bunch of files (.el/.elx, .png, .html and any additional files). This option is recommended for sending multiple files. You can turn it off for your convenience, if you send a single .el file (no screenshot, no XML, no additional files, etc).

Note: some send methods doesn't support sending multiple files, so this option may be required. For this reason it's recommended to keep it checked, until you're sure about send files count.

13. "**Add BugID to file name**" (.sndAddBugIDInFileName) option alters default file name by appending a Bug ID 421 value to it. Bug ID is a hash value of type and location of the problem. Exceptions with the same Bug ID is considered to be the same.

Use this option to generate a more customized/unique bug report's file names.

Example: "Project1_A5810000.elp".

14. "**Add client's computer name to file name**" (.sndAddComputerNameInFileName) option alters default file name by appending a name of client's machine value to it. All non-allowed file name characters will be replaced with safe replace character ('_').

Use this option to generate a more customized/unique bug report's file names.

Example: "Project1_Alex_Notebook.elp".

15. "**Add current date-time to file name**" (.sndAddDateInFileName) option alters default file name by appending a current date-time value to it ('yyyymmddhhnnss').

Use this option to generate a more customized/unique bug report's file names.

Example: "Project1_20110609005134.elp".

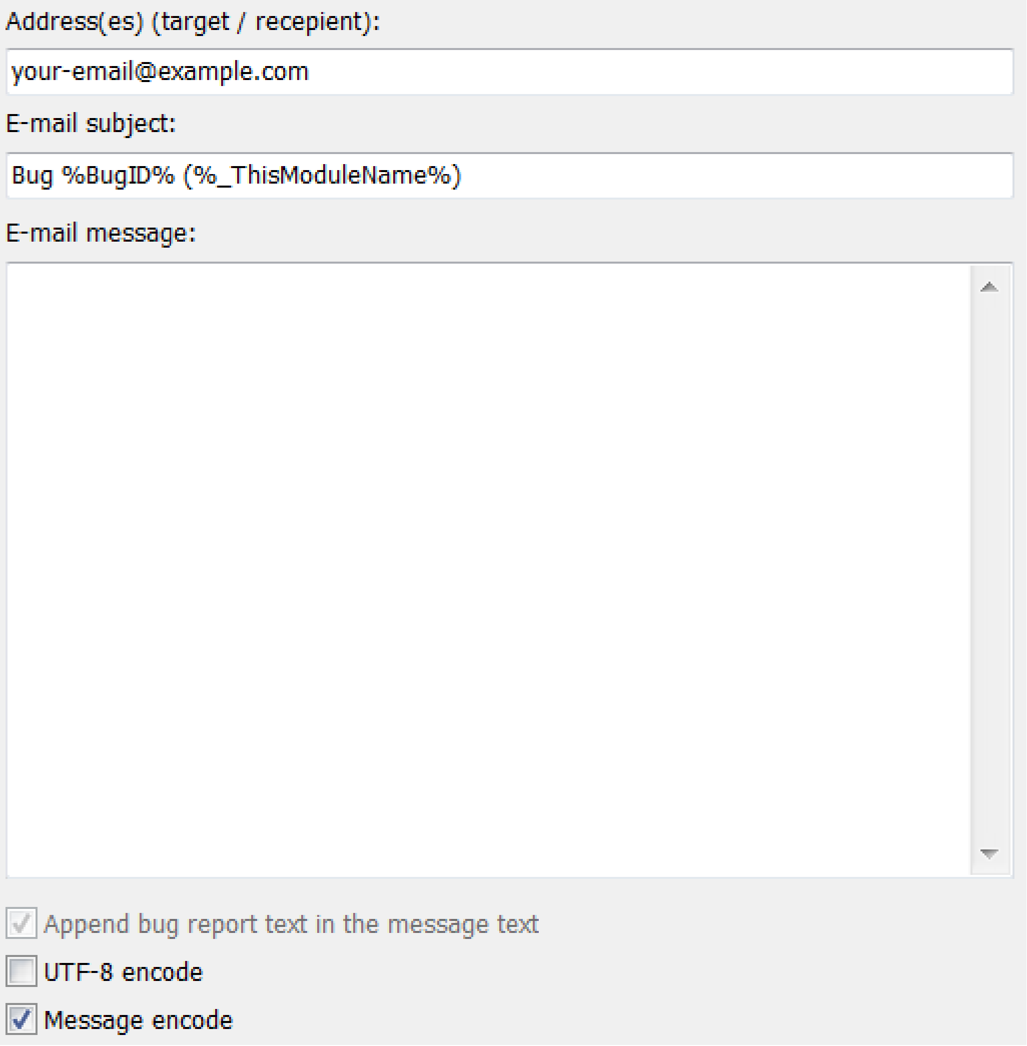
16. "**Pack with password**" (.ZipPassword) option encrypts .elp file (which is actually a ZIP archive) with specified password. Use this option to protect .elp files during sending. You don't need this option if you use SSL/TLS.

Notes:
- This option doesn't guarantee full protection, since password is stored inside your executable file. Even if it's encrypted - it's still stored inside .exe, so it can be stolen.
- It's not the same as bug report content's password.

See also:
- Report sending setup page 302
- Configuring sending report 53
- Bug report setup page 264
- OnAttachedFilesRequest
- OnZippedFilesRequest
- Security Considerations 158

10.3.5.2 Shell send

This is setup options for Shell send method 391 (also known as "mailto: protocol"; esmShellClient). They are located at Sending tab 302.



Address(es) (target / recepient):

your-email@example.com

E-mail subject:

Bug %BugID% (%_ThisModuleName%)

E-mail message:

☑ Append bug report text in the message text

□ UTF-8 encode

☑ Message encode

Shell send method options

1. "**Address(es)**" (`.SendShellTarget`) option specifies target e-mail address to send bug report to. Specify here your e-mail address for bug reports harvesting. Multiple e-mail addresses are allowed (separate them with "," or ";"), but this is usually not a good idea, since not every e-mail client software support this.

2. "**E-mail subject**" (`.SendShellSubject`) option specifies header (subject) for all sent bug reports. You can specify generic static text here (like 'Bug report for Project X') or use a `%tag%` to generate dynamic subject to distinguish one bug report from another. See using variables 228 for more info.

3. "**E-mail message**" (`.SendShellMessage`) option is optional text of e-mail message (body). You can enter here any text, use variables or just leave this field empty. Bug report text will appear here automatically.

4. "**Append bug report text in the message text**" option appends bug report text to the e-mail message (body). If e-mail body is empty - e-mail will consist of bug report text only. If you've entered non-null text - the first will be your text, the next will be bug report text.

This option is always checked for Shell send method, you can't turn it off - because this send method doesn't support attaching files, so there is no other way to send bug report, except inserting it into message.

5. "**Use ShellExecuteEx**" (`.SendShellUseShellExecute`) option defines which method should application use to run mailto links.

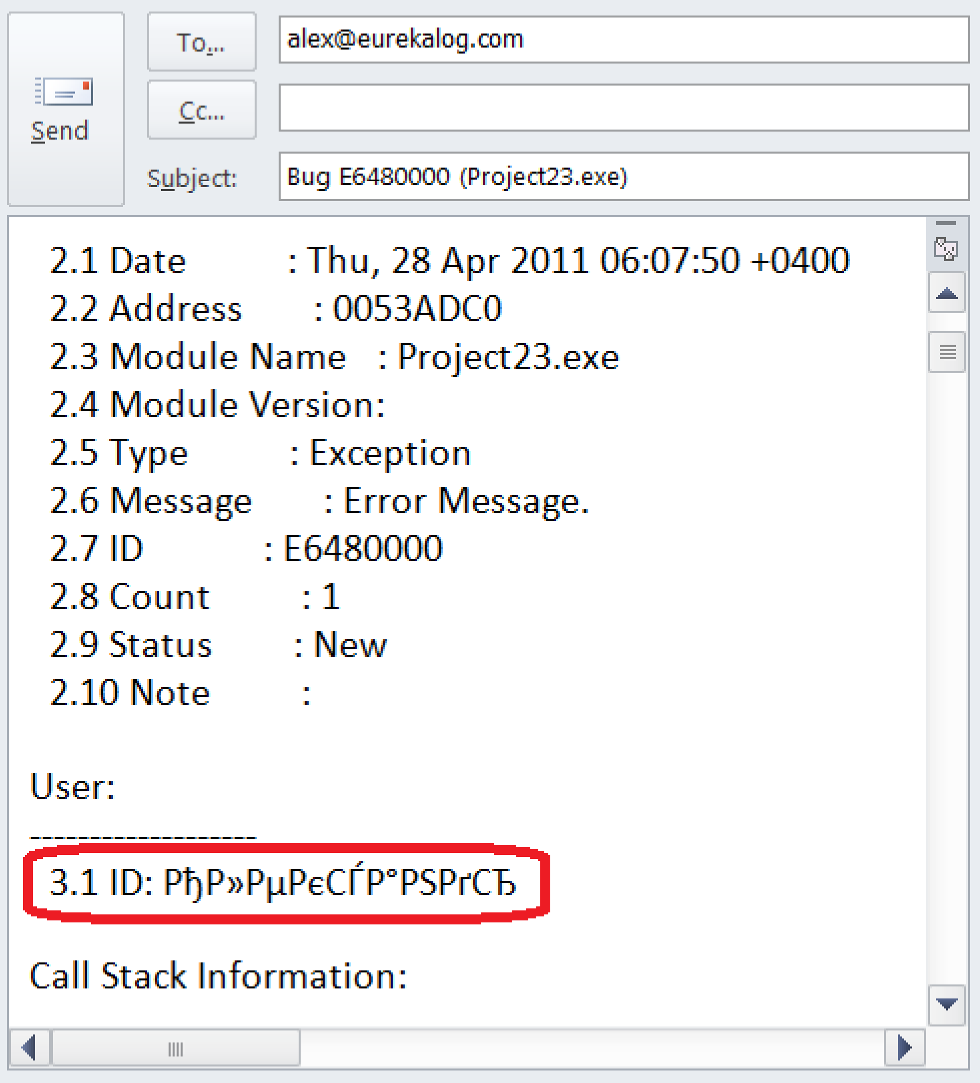
Checked: use ShellExecuteEx function to open mailto link.
Unchecked: use CreateProcess function to open mailto link.

`ShellExecuteEx` is restricted to `INTERNET_MAX_URL_LENGTH` (about 2048) characters.
`CreateProcess` is restricted to 32'767 characters.

However, when using `CreateProcess` function - application must manually resolve mailto protocol registration to obtain executable of mail application. This may or may not be the same application as used by `ShellExecuteEx` function.

Check this option to get maximum compatibility.
Uncheck this option to get maximum information length.

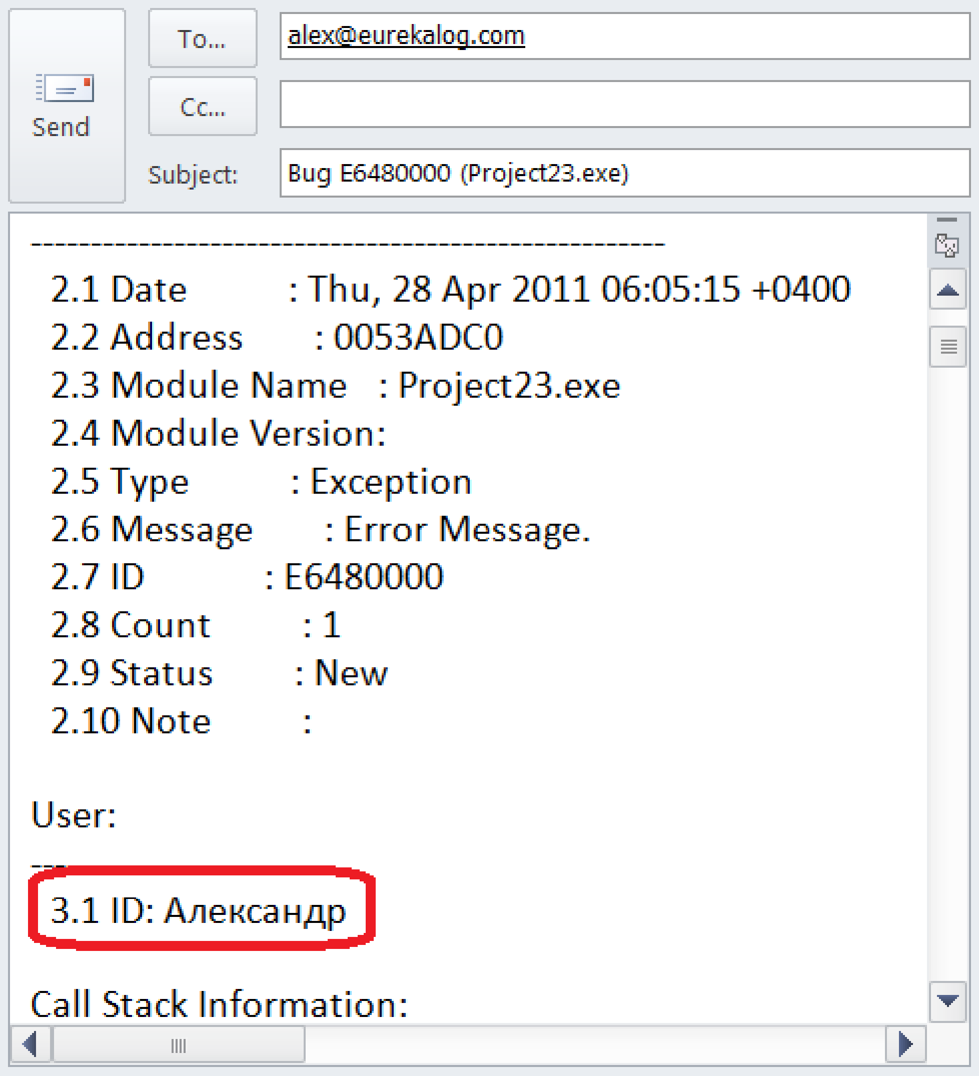
6. "**UTF-8 encode**" (`.SendShellUTF8`) option encodes subject and message in UTF-8 encoding (unicode). If this option is unchecked - these strings will be send as ANSI (in current ANSI encoding). Check this option to send extended characters outside of current ANSI page. Uncheck this option to use old and compatible ANSI encoding. The issue with that option is that not every e-mail client supports UTF-8 in mailto protocol. If e-mail client doesn't support UTF-8 and you've checked this option - resulting mail will contain non-English text in invalid encoding. For example:



To: alex@eurekalog.com
Subject: Bug E6480000 (Project23.exe)

```
  2.1 Date          : Thu, 28 Apr 2011 06:07:50 +0400
  2.2 Address       : 0053ADC0
  2.3 Module Name   : Project23.exe
  2.4 Module Version:
  2.5 Type          : Exception
  2.6 Message       : Error Message.
  2.7 ID          : E6480000
  2.8 Count         : 1
  2.9 Status        : New
  2.10 Note         :

User:
--------------------
  3.1 ID: РђР»РµРєСЃР°РЅРґСЂ

Call Stack Information:
```

Incorrect: "UTF-8" option is checked, but e-mail client doesn't support UTF-8



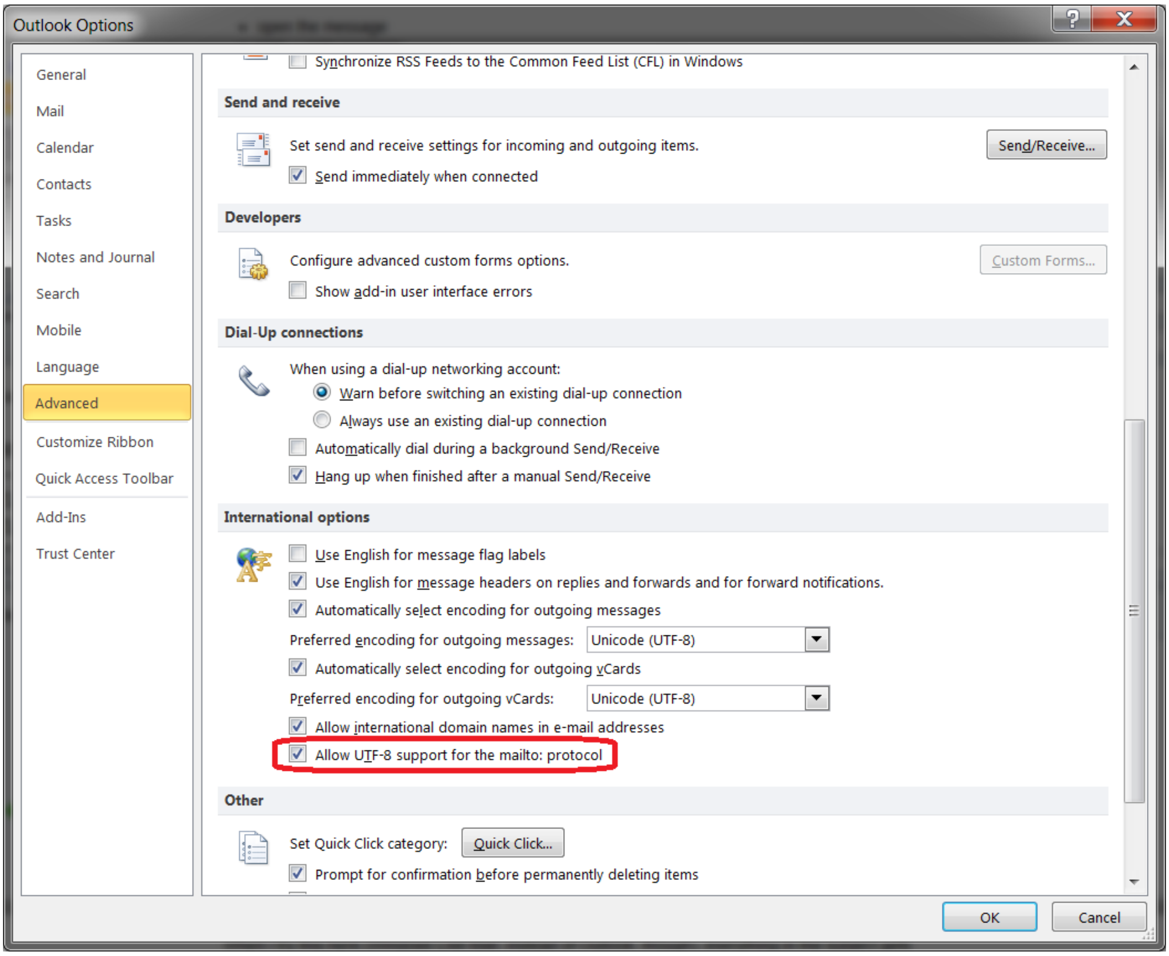
alex@eurekalog.com

Bug E6480000 (Project23.exe)

```
----------------------------------------------------
  2.1 Date          : Thu, 28 Apr 2011 06:05:15 +0400
  2.2 Address       : 0053ADC0
  2.3 Module Name   : Project23.exe
  2.4 Module Version:
  2.5 Type          : Exception
  2.6 Message       : Error Message.
  2.7 ID            : E6480000
  2.8 Count         : 1
  2.9 Status        : New
  2.10 Note         :

User:
---
  3.1 ID: Александр

Call Stack Information:
```

Correct: "UTF-8" option is checked and e-mail client supports UTF-8

Note: some e-mail clients may support or don't support UTF-8 in mailto protocol depending on their settings. For example, Outlook have "Allow UTF-8 support for mailto: protocol" option, which is located in "Advanced" section of Outlook's options:



□ Synchronize RSS Feeds to the Common Feed List (CFL) in Windows

Send and receive

Set send and receive settings for incoming and outgoing items.

☑ Send immediately when connected

Developers

Configure advanced custom forms options.

□ Show add-in user interface errors

Dial-Up connections

When using a dial-up networking account:

● Warn before switching an existing dial-up connection

○ Always use an existing dial-up connection

□ Automatically dial during a background Send/Receive

☑ Hang up when finished after a manual Send/Receive

International options

□ Use English for message flag labels

☑ Use English for message headers on replies and forwards and for forward notifications.

☑ Automatically select encoding for outgoing messages

Preferred encoding for outgoing messages: Unicode (UTF-8)

☑ Automatically select encoding for outgoing vCards

Preferred encoding for outgoing vCards: Unicode (UTF-8)

☑ Allow international domain names in e-mail addresses

☑ Allow UTF-8 support for the mailto: protocol

Other

Set Quick Click category:

☑ Prompt for confirmation before permanently deleting items

UTF-8 support option for the mailto: protocol in Outlook 2010

Note: enabling "UTF-8" option may decrease limit on maximum e-mail length due to multi-byte character encoding.

7. "**Message encode**" (.SendShellEncode) option enables %-encoding for some characters, as required by mailto protocol specification. You can uncheck this option, if you have some problems with e-mail clients.

Suppose you're opening URL "mailto:example@example.com&subject=Hello%20World". Internet Explorer decodes the URL, but the Windows "Run..." command does not. For example, if the link above is followed through Internet Explorer, the command line would be:

```
"C:\Program Files\EMailClient\client.exe" "mailto:example@example.com&subject=Hello World"
```

If this link is followed through Windows Explorer, the Windows Run command, or some other application, the command line would be:

```
"C:\Program Files\EMailClient\client.exe" "mailto:example@example.com&subject=Hello%20World"
```

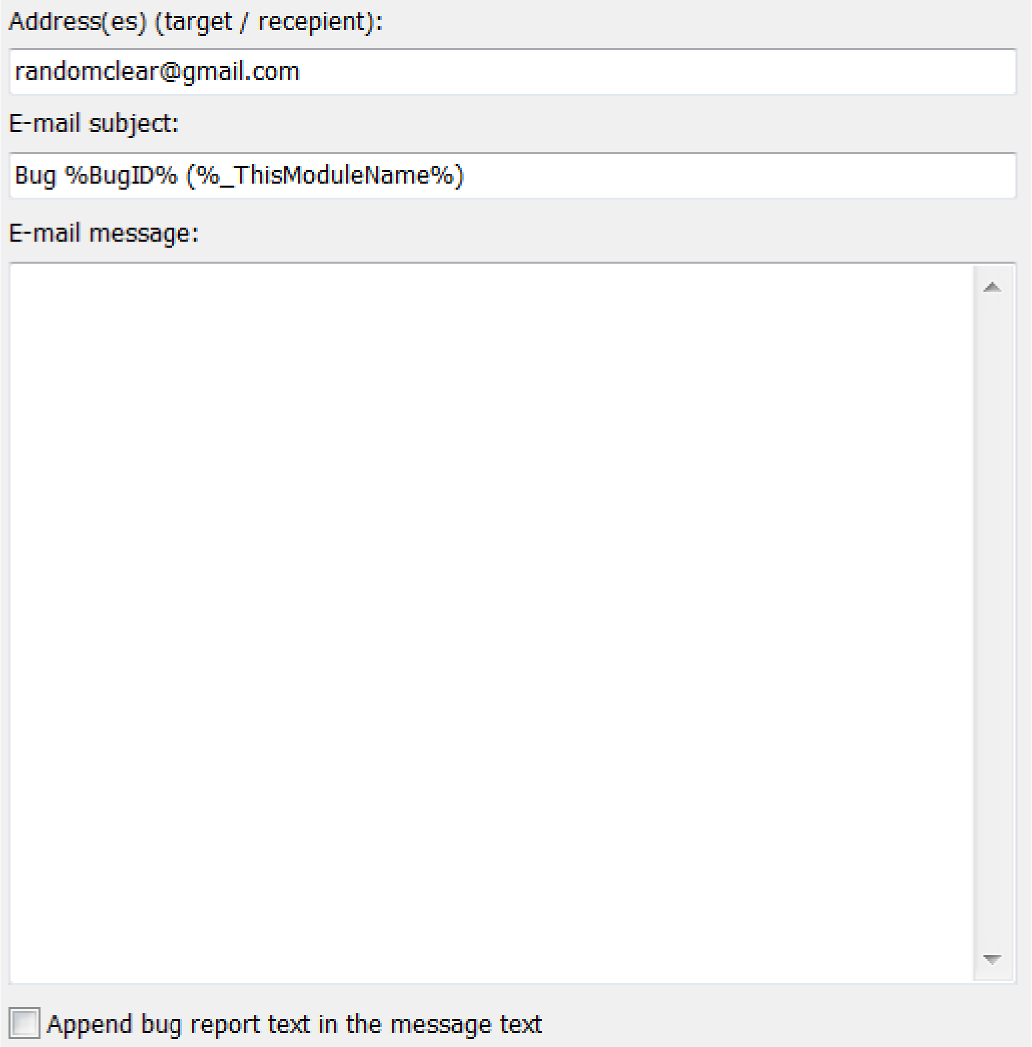
The "Message encode" option is used to switch between these two cases:
- "Message encode" is off: "mailto:example@example.com&subject=Hello World"
- "Message encode" is on: "mailto:example@example.com&subject=Hello%20World"

See also:
- Shell send method 391 for general description of this send method
- General send options 304

- Configuring send method 53
- Security Considerations 158

10.3.5.3 Simple MAPI

This is setup options for Simple MAPI send method 393 (esmSimpleMAPI). They are located at Sending tab 302.



Address(es) (target / recepient): randomclear@gmail.com

E-mail subject: Bug %BugID% (%_ThisModuleName%)

E-mail message:

□ Append bug report text in the message text

Simple MAPI send method options

1. "**Address(es)**" (.SendSMAPITarget) option specifies target e-mail address to send bug report to. Specify here your e-mail address for bug reports harvesting. Multiple e-mail addresses are allowed (separate them with "," or ";").

2. "**E-mail subject**" (.SendSMAPISubject) option specifies header (subject) for all sent bug reports. You can specify generic static text here (like 'Bug report for Project X') or use a %tag% to generate dynamic subject to distinguish one bug report from another. See using variables 228 for more info.

3. "**E-mail message**" (.SendSMAPIMessage) option is optional text of e-mail message (body). You can enter here any text, use variables or just leave this field empty.

4. "**Append bug report text in the message text**" (.SendSMAPIAppendLogs) option appends bug report text to the e-mail message (body). If e-mail body is empty - e-mail will consist of bug report text only. If you've entered non-null text - the first will be your text, the next will be bug report text.

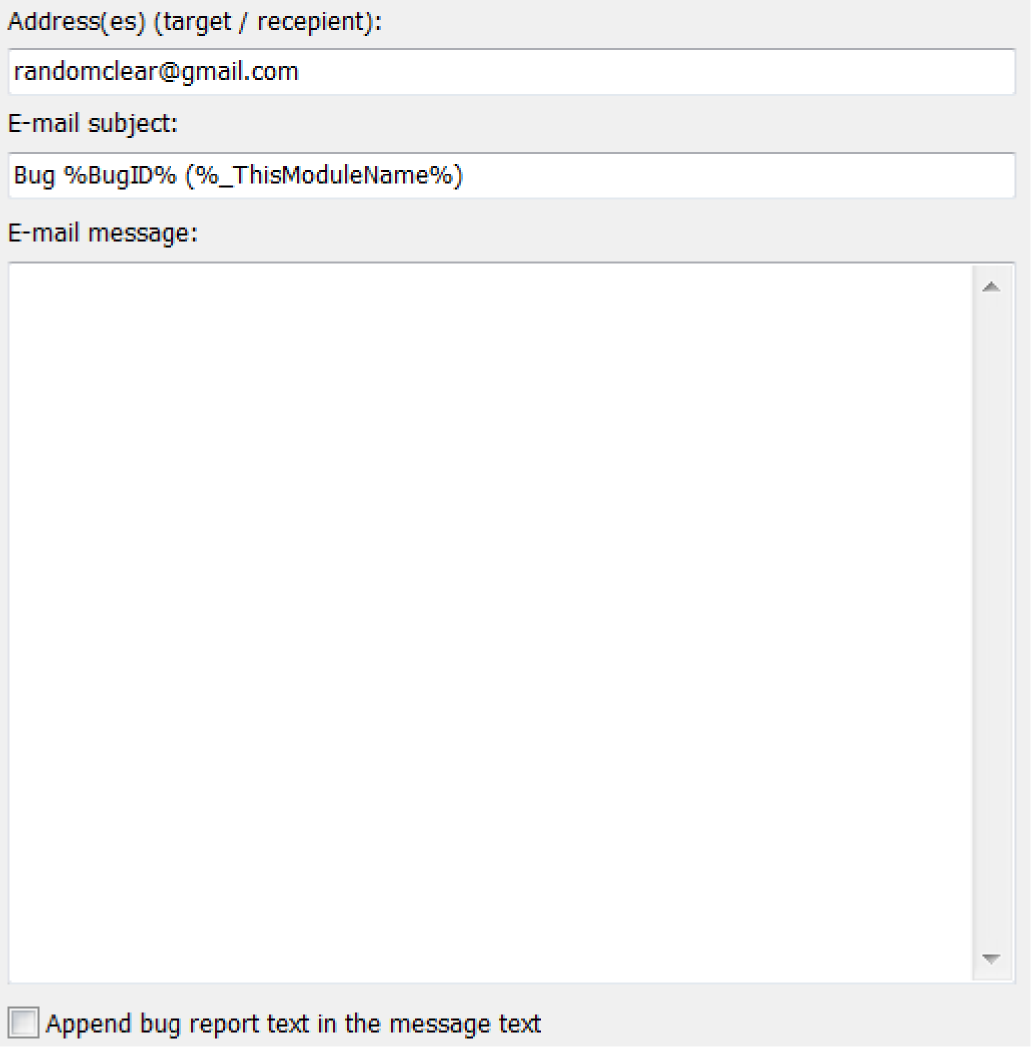
Checking this option will remove bug report file from attached files. Any other separately attached files (if any) are not affected. Unchecking this option will result into attaching bug report file as standard attach.

See also:

- Simple MAPI send method 393 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158

10.3.5.4 MAPI

This is setup options for MAPI send method 396 (`esmMAPI`). They are located at Sending tab 302.



Address(es) (target / recepient):

randomclear@gmail.com

E-mail subject:

Bug %BugID% (%_ThisModuleName%)

E-mail message:

□ Append bug report text in the message text

Simple MAPI send method options

1. "**Address(es)**" (`.SendMAPITarget`) option specifies target e-mail address to send bug report to. Specify here your e-mail address for bug reports harvesting. Multiple e-mail addresses are allowed (separate them with "," or ";").

2. "**E-mail subject**" (`.SendMAPISubject`) option specifies header (subject) for all sent bug reports. You can specify generic static text here (like 'Bug report for Project X') or use a `%tag%` to generate dynamic subject to distinguish one bug report from another. See using variables 228 for more info.

3. "**E-mail message**" (.SendMAPIMessage) option is optional text of e-mail message (body). You can enter here any text, use variables or just leave this field empty.

4. "**Append bug report text in the message text**" (.SendMAPIAppendLogs) option appends bug report text to the e-mail message (body). If e-mail body is empty - e-mail will consist of bug report text only. If you've entered non-null text - the first will be your text, the next will be bug report text.

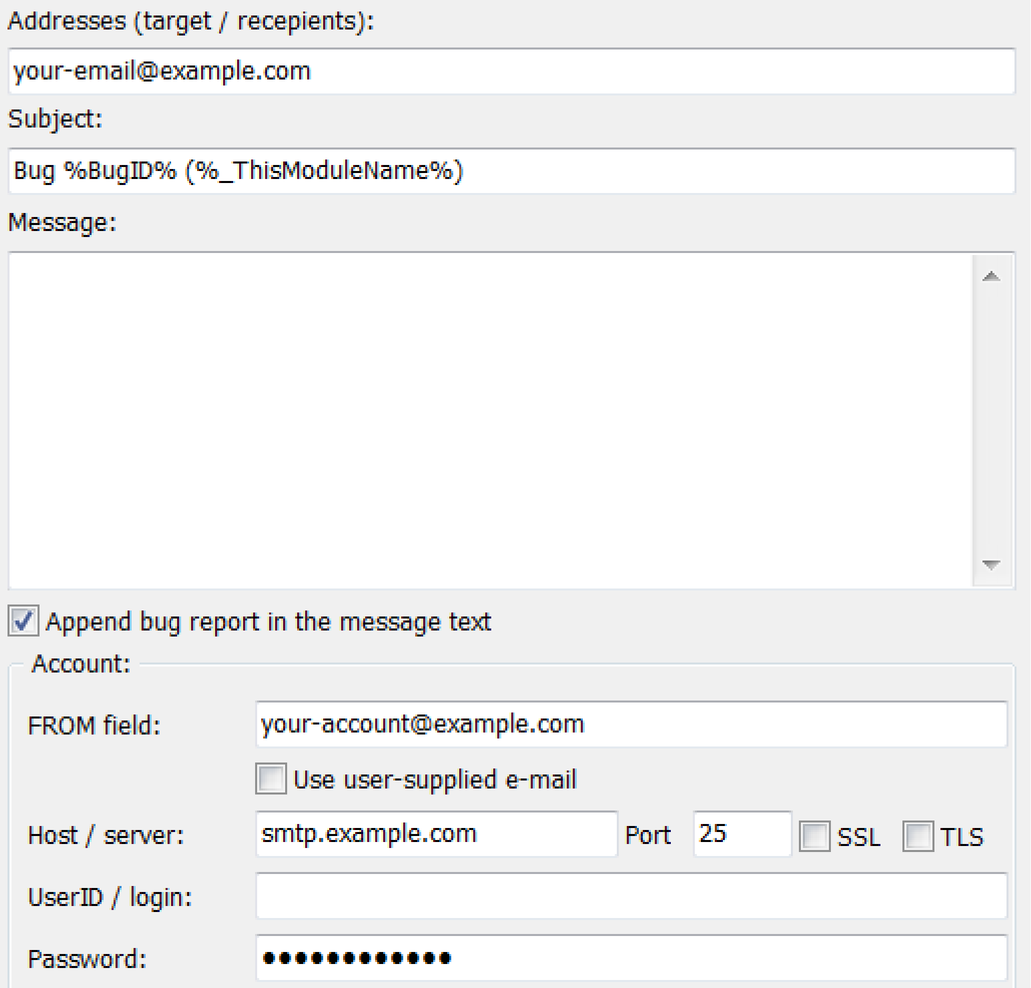
Checking this option will remove bug report file from attached files. Any other separately attached files (if any) are not affected. Unchecking this option will result into attaching bug report file as standard attach.

See also:

- MAPI send method 398 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158

10.3.5.5 SMTP client

This is setup options for SMTP client send method 397 (esmSMTPClient). They are located at Sending tab 302.



Addresses (target / recepients):

your-email@example.com

Subject:

Bug %BugID% (%_ThisModuleName%)

Message:

☑ Append bug report in the message text

Account:

FROM field: your-account@example.com

□ Use user-supplied e-mail

Host / server: smtp.example.com Port 25 □ SSL □ TLS

UserID / login:

Password: ●●●●●●●●●●●●●

SMTP client send method options

1. "**Address(es)**" (.SendSMTPClientTarget) option specifies target e-mail address to send bug report to. Specify here your e-mail address for bug reports harvesting. Multiple e-mail addresses are allowed (separate them with "," or ";").

2. "**E-mail subject**" (`.SendSMTPClientSubject`) option specifies header (subject) for all sent bug reports. You can specify generic static text here (like 'Bug report for Project X') or use a `%tag%` to generate dynamic subject to distinguish one bug report from another. See using variables 228 for more info.

3. "**E-mail message**" (`.SendSMTPClientMessage`) option is optional text of e-mail message (body). You can enter here any text, use variables or just leave this field empty.

4. "**Append bug report text in the message text**" (`.SendSMTPClientAppendLogs`) option appends bug report text to the e-mail message (body). If e-mail body is empty - e-mail will consist of bug report text only. If you've entered non-null text - the first will be your text, the next will be bug report text.

Checking this option will remove bug report file from attached files. Any other separately attached files (if any) are not affected. Unchecking this option will result into attaching bug report file as standard attach.

5. "**From field**" (`.SendSMTPClientFrom`) option specifies your e-mail address. It's your real e-mail account on e-mail server, which will be used to send bug reports. It can be the same as "Address(es)" option, but it doesn't have to.

6. "**Use user-supplied e-mail**" (`.SendSMTPClientUseRealEMail`) option allows you to substitute your real e-mail address with customer's e-mail (which can be set via some error dialogs 370 or SetUserEMail function). If you turn this option on - you will see customer's e-mail in FROM field in bug reports. This is convenient. But usually you need to keep this option unchecked, since most e-mail servers will not allow you to send e-mails as from other people.

7. "**Host / server**" (`.SendSMTPClientHost`) option specifies e-mail server to use. Please, refer to your e-mail server's support/help to get this value. Usually, if you have *account@domain.com*, then this value could be *smtp.domain.com*, *mail.domain.com* or *mx.domain.com*.

8. "**Port**" (`.SendSMTPClientPort`) option specified TCP port number. Again, refer to your e-mail server's support/help to get this value. Typical values are 25, 587 and 465.

7. "**SSL**" (`.SendSMTPClientSSL`) and "**TLS**" (`.SendSMTPClientTLS`) options enabled secure mode for e-mail server. Check one of these options only if your e-mail server requires it. Please, see this article 588 to know more about these mode differences. Usually, it's best to turn on "TLS" checkbox, even if your e-mail server doesn't require it.

8. "**UserID / login**" (`.SendSMTPClientLogin`) option specifies your login on e-mail server. Usually, it's the same as your e-mail or part of it before @. For example, if you have *account@domain.com*, then your login will be either *account* or *account@domain.com*.

9. "**Password**" (`.SendSMTPClientPassword`) option is your password on e-mail server. Currently EurekaLog supports AUTH LOGIN and AUTH PLAIN authentication methods.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use your personal e-mail for this. Create a new special account for bug reporting via this method (and be sure to protect it against e-mail change or hi-jacking).

Examples of setup for different common e-mail servers

GMail
FROM field: *your-account*@gmail.com or *your-account*@*your-domain* (for customized GMail accounts)
Host / server: smtp.gmail.com
Port: 587
SSL: False

TLS: True
Login: *your-account*@gmail.com or *your-account*@*your-domain* (the same as "FROM field")
Password: *your-password*

Notes:

- Port 25 will not work.
- Alternatively, you can use port 465 with SSL enabled (TLS disabled).
- Account name (i.e. without domain part) sometimes can be accepted as login, but we recommend to specify full e-mail address as login.
- You must enter "*application password*" instead of account password, if you're using two-factor authentication. **We strongly recommend to enable this additional protection**, if you use SMTP client mode in your applications.

See also: Configuring other mail clients and Security Considerations 158.

HotMail (Microsoft Live)
FROM field: *your-account*@hotmail.com or *your-account*@live.com or *your-account*@msn.com
Host / server: smtp.live.com
Port: 587
SSL: False
TLS: True
Login: *your-account*@hotmail.com or *your-account*@live.com or *your-account*@msn.com (the same as "FROM field")
Password: *your-password*

Alternatively, you can use port 25 with the rest of the settings to be the same.

See also: Problems with access to Hotmail from other e-mail clients (specifically: Send and receive Windows Live Hotmail e-mails with mail clients).

Yahoo!
FROM field: *your-account*@yahoo.com
Host / server: smtp.mail.yahoo.com
Port: 465
SSL: True
TLS: False
Login: *your-account*
Password: *your-password*

Note: you need **Yahoo! Mail Plus** account to use external e-mail client.

See also: How to access Yahoo! Mail Plus using an email reader.

AOL / AIM
FROM field: *your-account*@aol.com or *your-account*@aim.com
Host / server: smtp.aol.com or smtp.aim.com (depending on your account type)
Port: 587
SSL: False
TLS: True
Login: *your-account*
Password: *your-password*

See also: Read and Send AOL/AIM E-mail with Other E-mail Applications.

Note: some e-mail servers with web UI requires you to explicitly allow access to mail from 3rd party clients in e-mail account settings. Please, refer to help or support services of your e-mail server.

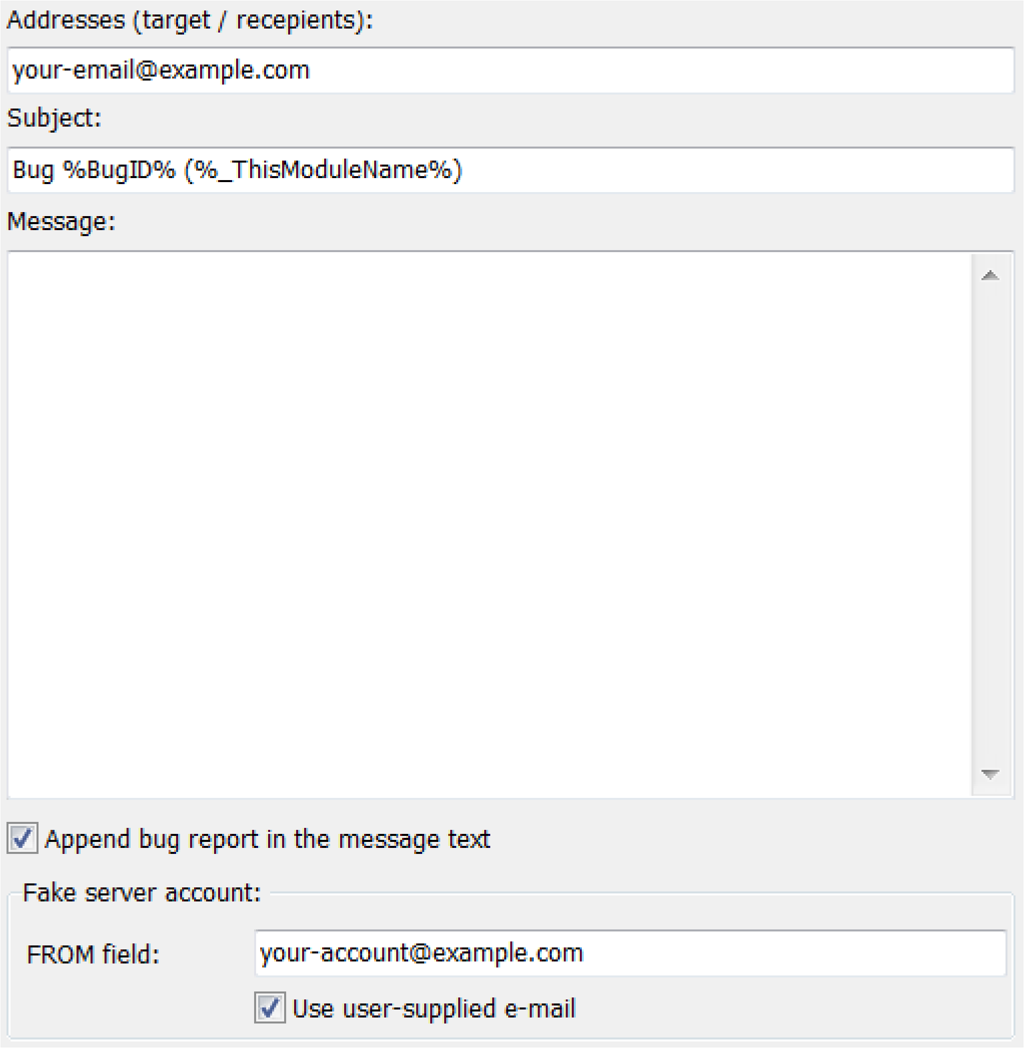
See also:

- SMTP client send method 397 for general description of this send method
- General send options 304
- Differences between SSL and TLS modes 588
- Configuring send method 53

- Security Considerations 158

10.3.5.6 SMTP server

This is setup options for SMTP server send method 398 (esmSMTPServer). They are located at Sending tab 302.



Addresses (target / recepients):

your-email@example.com

Subject:

Bug %BugID% (%_ThisModuleName%)

Message:

☑ Append bug report in the message text

Fake server account:

FROM field: your-account@example.com

☑ Use user-supplied e-mail

SMTP server send method options

1. "**Address(es)**" (.SendSMTPServerTarget) option specifies target e-mail address to send bug report to. Specify here your e-mail address for bug reports harvesting. Multiple e-mail addresses are allowed (separate them with "," or ";").

2. "**E-mail subject**" (.SendSMTPServerSubject) option specifies header (subject) for all sent bug reports. You can specify generic static text here (like 'Bug report for Project X') or use a %tag% to generate dynamic subject to distinguish one bug report from another. See using variables 228 for more info.

3. "**E-mail message**" (.SendSMTPServerMessage) option is optional text of e-mail message (body). You can enter here any text, use variables or just leave this field empty.

4. "**Append bug report text in the message text**" (.SendSMTPServerAppendLogs) option appends bug report text to the e-mail message (body). If e-mail body is empty - e-mail will consist of bug report text only. If you've entered non-null text - the first will be your text, the next will be bug report text.

Checking this option will remove bug report file from attached files. Any other separately attached files (if any) are not affected. Unchecking this option will result into attaching bug report file as standard attach.

5. "**From field**" (`.SendSMTPServerFrom`) option specifies your e-mail address. It's can be any valid e-mail address, which will be used as "sender" to send bug reports. It can be the same as "Address(es)" option, but it doesn't have to. This doesn't have to be real e-mail account. It can be anything.

6. "**Use user-supplied e-mail**" (`.SendSMTPServerUseRealEMail`) option allows you to substitute your real e-mail address with customer's e-mail (which can be set via some error dialogs 370 or SetUserEMail function). If you turn this option on - you will see customer's e-mail in FROM field in bug reports. This is convenient. Usually, you want to keep this option checked.

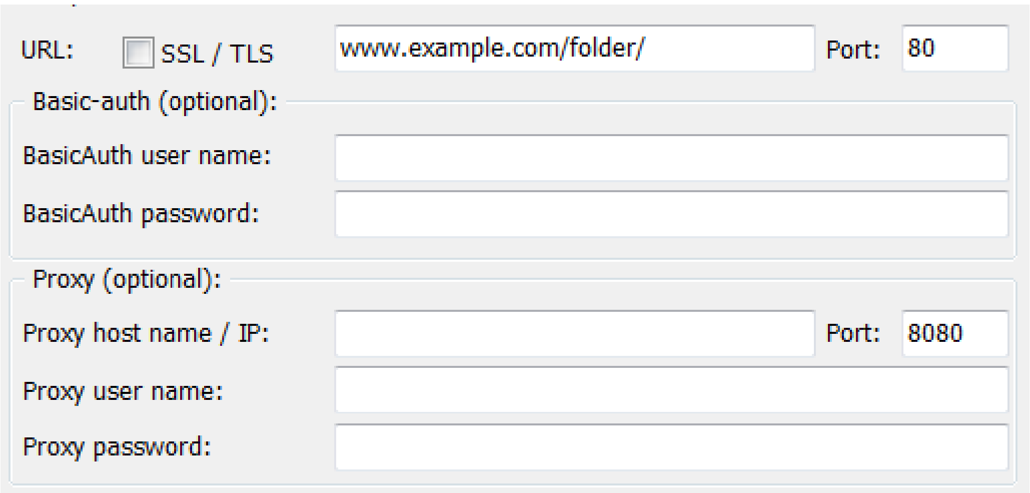
Important Note: you can use `CurrentEurekaLogOptions.CustomHELO` option to alter name to be passed to `HELO`/`EHLO` SMTP command.

See also:
- SMTP server send method 398 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158

10.3.5.7 HTTP upload

This is setup options for HTTP upload send method 398 (`wsmHTTP`). They are located at Sending tab 302.

Note: HTTP upload can be used as received for bug tracker. See example for FogBugz 115.



URL: □ SSL / TLS www.example.com/folder/ Port: 80

Basic-auth (optional):

BasicAuth user name:

BasicAuth password:

Proxy (optional):

Proxy host name / IP: Port: 8080

Proxy user name:

Proxy password:

HTTP upload send method options

1. "**URL**" (`.SendHTTPURL`) option specifies target URL for file upload. Usually, it's script name on web server. See description of HTTP upload method 398 for discussion of upload scripts. Do not add here "http://" or ":80" parts here. Specify only domain name (or IP), path and script name. Example: `www.example.com/folder/upload.php`

Warning: be sure to setup adequate maximum upload file limits in your web-server/script configuration. Otherwise sending may fail on large bug reports.

2. "**Port**" (`.SendHTTPPort`) option specifies HTTP port on web server. It's 80 by default. Other common value is 8080. For SSL/TLS it's usually 443.

3. "**SSL / TLS**" (`.SendHTTPSSL`) option enabled secure mode (HTTPS protocol). Don't forget to adjust port number, if you change this checkbox.

4. "**BasicAuth**" (`.SendHTTPAuthLogin`, `.SendHTTPAuthPassword`) options specify account details for Basic-Auth type authentication. Usually these fields are blank. Fill them, if you use Basic-Auth on your web server.

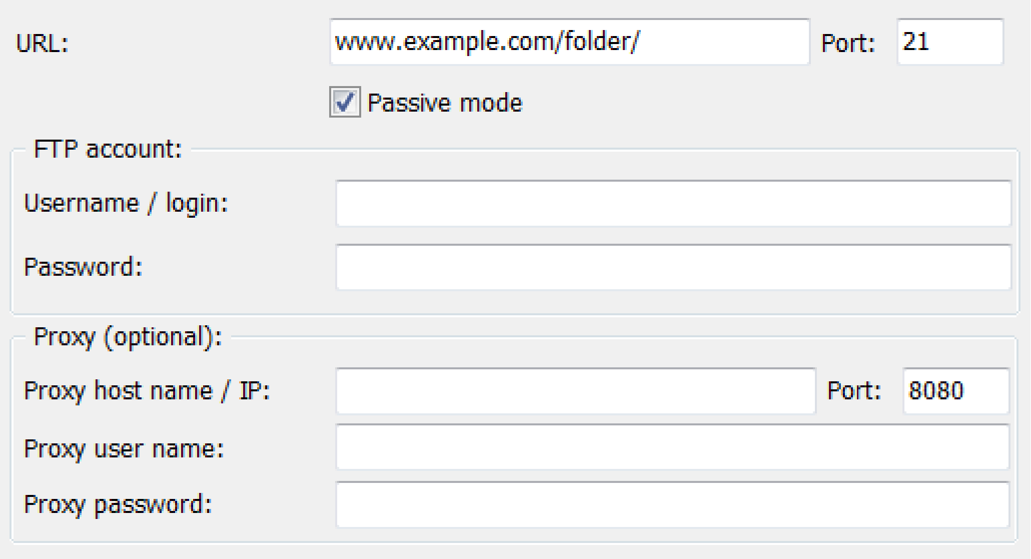
Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. Create a new special account for bug reporting via this method.

5.
"**Proxy**" (`.SendHTTPProxyHost`, `.SendHTTPProxyPort`, `.SendHTTPProxyLogin`, `.SendHTTPProxyPassword`) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

See also:
- HTTP upload send method 398 for general description of this send method
- General send options 304
- Configuring send method 53
- HTTP upload setup for FogBugz 115
- Security Considerations 158

10.3.5.8 FTP upload

This is setup options for FTP upload send method 404 (`wsmFTP`). They are located at Sending tab 302.


URL: www.example.com/folder/ Port: 21

☑ Passive mode

FTP account:

Username / login:

Password:

Proxy (optional):

Proxy host name / IP: Port: 8080

Proxy user name:

Proxy password:

FTP upload send method options

1. "**URL**" (`.SendFTPURL`) option specifies target URL for file upload. Usually, folder on file server. Do not add here "ftp://" or ":21" parts here. Specify only domain name (or IP) and path. Example: `www.example.com/folder/`

2. "**Port**" (`.SendFTPPort`) option specifies FTP port on file server. It's 21 by default.

3. "**Passive mode**" (`.SendFTPPassiveMode`) option enables so-called "FTP passive mode", which is more friendly to client's firewall/NAT/network configuration than standard (active) mode.

FTP can be run in active or passive mode, which determine how the data connection is established. In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection. This is a default standard mode. However, in situations where the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode the client sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server. It's recommended to enable passive mode to bypass client's NAT/firewall.

4. "**FTP account**" (`.SendFTPLogin`, `.SendFTPPassword`) options specify your account details on FTP server.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. Create a new special account for bug reporting via this method.

Note: EurekaLog doesn't support SFTP protocol.

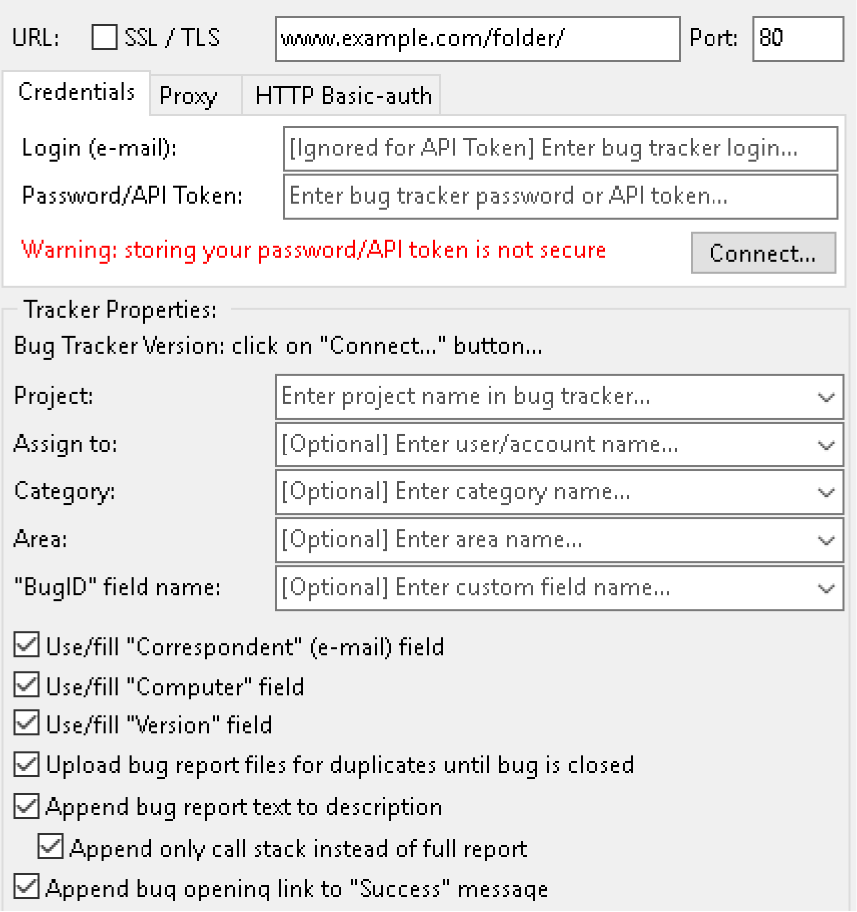
5.
"**Proxy**" (`.SendFTPProxyHost`, `.SendFTPProxyPort`, `.SendFTPProxyLogin`, `.SendFTPProxyPassword`) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

See also:
- FTP upload send method 404 for general description of this send method
- General send options 304
- Differences between "SSL mode" and "TLS mode" 588
- Configuring send method 53
- Security Considerations 158

10.3.5.9 FogBugz

This is setup options for FogBugz send method 404 (`wsmFogBugz`). They are located at Sending tab 302.

Note: you may consider using HTTP upload method 398 for FogBugz instead of using FogBugz API. See FogBugz setup 108 for more detailed description; see FogBugz: using HTTP upload 115 for detailed manual on HTTP Upload setup for FogBugz.



URL: □ SSL / TLS www.example.com/folder/ Port: 80

Login (e-mail): [Ignored for API Token] Enter bug tracker login...

Password/API Token: Enter bug tracker password or API token...

Warning: storing your password/API token is not secure

Tracker Properties:

Bug Tracker Version: click on "Connect..." button...

Project: Enter project name in bug tracker...

Assign to: [Optional] Enter user/account name...

Category: [Optional] Enter category name...

Area: [Optional] Enter area name...

"BugID" field name: [Optional] Enter custom field name...

☑ Use/fill "Correspondent" (e-mail) field

☑ Use/fill "Computer" field

☑ Use/fill "Version" field

☑ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

FogBugz send method options

1. "**URL**" (.SendFogBugzURL) option specifies target URL for your ForBugz installation. Do not add here "http://" or ":80" parts here. Specify only domain name (or IP) and path. Example: `www.example.com/fogbugz/` or `account.fogbugz.com`

Warning: be sure to setup adequate maximum upload file limits in your web-server/ FogBugz configuration. Otherwise sending may fail on large bug reports.

2. "**Port**" (.SendFogBugzPort) option specifies HTTP port on web server. It's 80 by default. Other common value is 8080. For SSL/TLS it's usually 443. Port will be set automatically to 80/443 by default.

3. "**SSL / TLS**" (.SendFogBugzSSL) option enabled secure mode (HTTPS protocol). Port will be set automatically to 80/443 by default. Don't forget to adjust port number, if you are using alternative port number.

4. "**Login**" (.SendFogBugzLogin) and "**Password**" (.SendFogBugzPassword) options specify your account on FogBugz server. This account will be used to submit bug reports. You can use API token instead of password - leave login field blank in this case.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use FogBugz admin account here. Create a new special account for bug reporting via this method. Limit its rights to submitting only. As alternative - you can always use anonymous submission or BugzScout 115 - that way you will not store any credentials in your application.

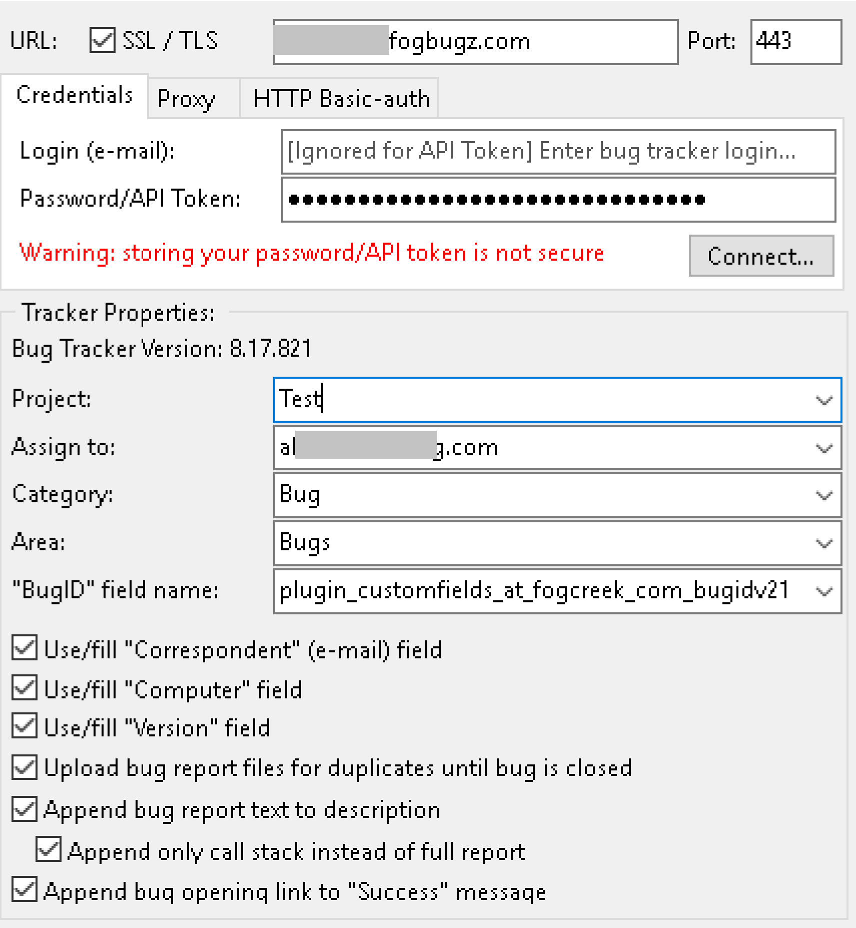
Note: it is a good idea to disable e-mail notifications for this account.

5. "**Proxy**" (.SendFogBugzProxyHost

, .SendFogBugzProxyPort, .SendFogBugzProxyLogin, .SendFogBugzProxyPassword) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

6. "**BasicAuth**" (.SendFogBugzBasicAuthLogin, .SendFogBugzBasicAuthPassword) options specify account details for Basic-Auth type authentication. Usually these fields are blank. Fill them, if you use Basic-Auth on your web server.

7. "**Connect**" button will try to connect to your FogBugz server using the specified URL/port/credentials. If you made mistake in your configuration - an error message will be displayed. In case of success - field below will be populated from configuration of your bug tracker.



URL: ☑ SSL / TLS [redacted]fogbugz.com Port: 443

Credentials | Proxy | HTTP Basic-auth

Login (e-mail): [Ignored for API Token] Enter bug tracker login...

Password/API Token: ••••••••••••••••••••••••••••••••••••

Warning: storing your password/API token is not secure

Connect...

Tracker Properties:

Bug Tracker Version: 8.17.821

Project: Test

Assign to: al[redacted]g.com

Category: Bug

Area: Bugs

"BugID" field name: plugin_customfields_at_fogcreek_com_bugidv21

☑ Use/fill "Correspondent" (e-mail) field

☑ Use/fill "Computer" field

☑ Use/fill "Version" field

☑ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

8. "**Project**" (.SendFogBugzProject) options specifies project name to store bug reports. It's mandatory.

9. "**Assign to**" (.SendFogBugzOwner) option specifies owner account name. If this option is empty, all submitted bug reports will be assigned to default account (which is usually a submitter account). If you enter here any account name (it could be the same as "Login" option) - all submitted bug reports will be assigned to this account.

Note: this option is ignored, if submitted bug report already exists (submitting a known issue).

10. "**Category**" (.SendFogBugzCategory) option specifies category for submitted reports. If this option is empty, all submitted bug report will belong to default category (which is usually "Bug" category). If you enter any category name here - all submitted bug reports will belong to the specified category.

11. "**Area**" (.SendFogBugzArea) option specifies area for submitted reports. If this option is empty, all submitted bug report will be assigned to default area (which is usually "Misc" area). If you enter any area name here - all submitted bug reports will be assigned to the

specified area.

12. "**BugID field name**" (`.SendFogBugzBugIDFieldName`) option specified name of custom field which EurekaLog can use to store BugID. While name can be a "user-friendly"-name (such as 'Bug ID') - we strongly recommend to use "internal"-name (such as '`plugin_customfields_at_fogcreek_com_bugidv21`'). Leave empty if you don't need standalone place to store BugID. It is purely optional, as EurekaLog will use internal BugzScout hash for merging.

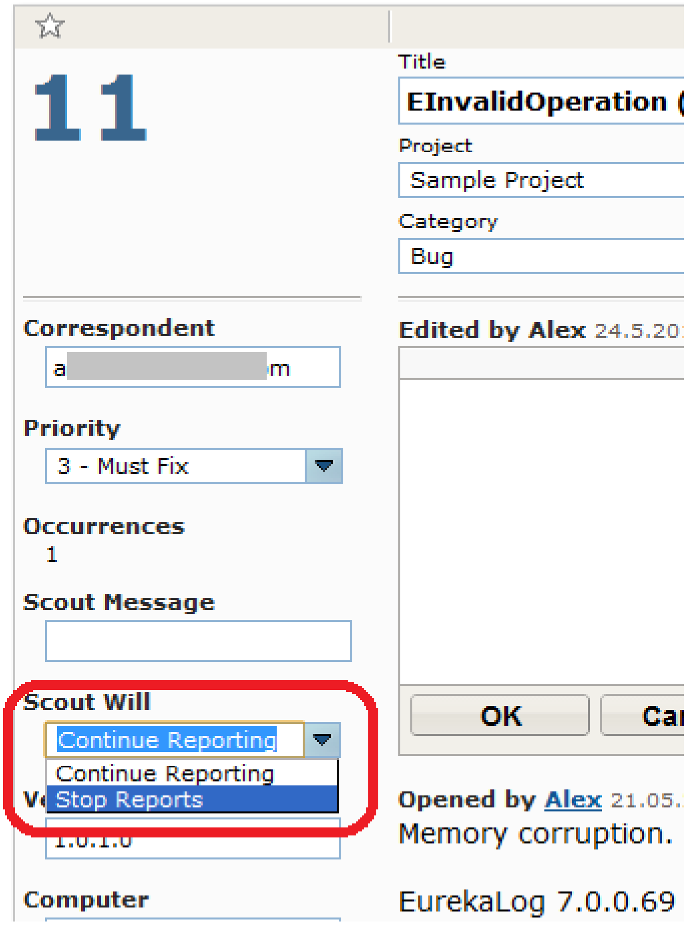
13. "**Use e-mail fields**" (`.SendFogBugzUseEMail`) option enables storing client's e-mail as "Correspondent" in ticket.

14. "**Use computer field**" (`.SendFogBugzUseComputer`) option instructs EurekaLog to use first custom field in FogBugz: "Computer". Disable this option, if you've renamed this field and/or use it for other purposes.

15. "**Use version field**" (`.SendFogBugzUseVersion`) option instructs EurekaLog to use second custom field in FogBugz: "Version". Disable this option, if you've renamed this field and/or use it for other purposes.

16. "**Upload bug report files for duplicates until bug is closed**" (`.SendFogBugzUploadFilesForDups`) option allows you to collect all bug reports. If this option is unchecked (default): only first bug report is uploaded and stored. All other bug reports for the same problem (identified by BugID) will be discarded. Only "Occurrences" field will be increased. If this option is checked: bug reports for the same problem will be uploaded to issue.

Notes:
- if you check this option be sure to have a lot of hard disk space to store all bug reports.
- bug reports will not be uploaded, once the issue is closed or resolved.
- you can disable bug report collection at any time, using BugzScout options of the issue:



11

Title
EInvalidOperation (

Project
Sample Project

Category
Bug

Correspondent
a[redacted]m

Priority
3 - Must Fix

Occurrences
1

Scout Message

Scout Will
Continue Reporting
Continue Reporting
Stop Reports

V
1.0.1.0

Computer

Edited by Alex 24.5.20

Opened by Alex 21.05.
Memory corruption.

EurekaLog 7.0.0.69

Disabling bug report collecting for issue

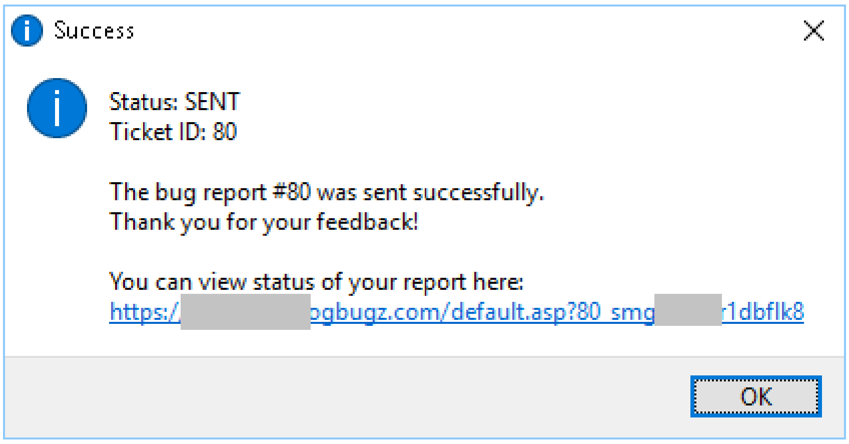
17. "**Append bug report text to description**" (`.SendFogBugzAppendText`) option allows you to insert bug report's text into "Description" field. It's convenient, if you need to peek bug report without downloading bug report file. You can turn this option off, if you don't need this behaviour.

Note: checking this option will not disable bug report file upload. File will still be attached.

18. "**Append only call stack instead of full report**" (`.SendFogBugzUseCallStackAsBugReport`) option alters previous option. Disabled: full bug report text will be added (e.g. general section, call stack, modules, processes, CPU/assembler, etc.); Enabled: only call stack will be added (you will still be able to view full bug report by downloading file attach).

19. "**Append bug opening link to "Success" message**" (`.SendFogBugzAllowLinks`) option will add a link to view bug report on FogBugz to message dialog after successful send. So end user (client) will be able to view status of the report on your bug tracker (read-only limited access).

This option has no effect if successful message dialog is disabled. Turn this option on for public bug trackers. Turn this option off for private bug trackers.

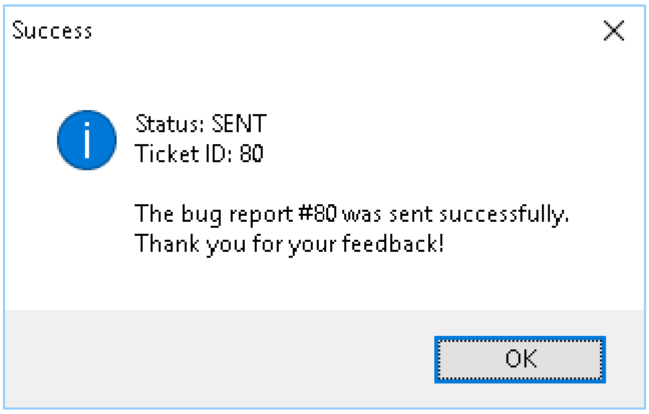


Status: SENT
Ticket ID: 80

The bug report #80 was sent successfully.
Thank you for your feedback!

You can view status of your report here:
https:/[redacted]ogbugz.com/default.asp?80_smg[redacted]r1dbflk8

"Append bug opening link to "Success" message" option is enabled



Status: SENT
Ticket ID: 80

The bug report #80 was sent successfully.
Thank you for your feedback!

"Append bug opening link to "Success" message" option is disabled

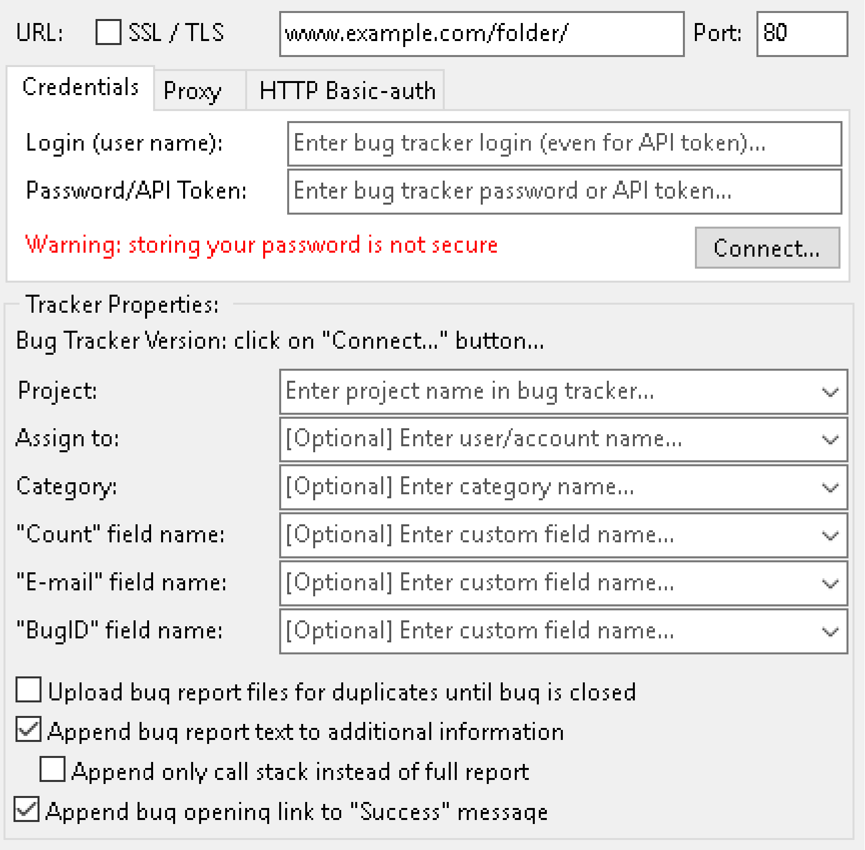
Note: active hyper-link will work on Windows Vista or later. It will be displayed as plain text on Windows XP and earlier.

See also:

- FogBugz send method 404 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158
- Managing bug reports in FogBugz 105
- FogBugs setup 108
- Using HTTP upload with FogBugz 115

10.3.5.10 Mantis

This is setup options for Mantis send method 406 (wsmMantis). They are located at Sending tab 302.



URL: □ SSL / TLS www.example.com/folder/ Port: 80

Credentials | Proxy | HTTP Basic-auth

Login (user name): Enter bug tracker login (even for API token)...

Password/API Token: Enter bug tracker password or API token...

Warning: storing your password is not secure

Connect...

Tracker Properties:

Bug Tracker Version: click on "Connect..." button...

Project: Enter project name in bug tracker...

Assign to: [Optional] Enter user/account name...

Category: [Optional] Enter category name...

"Count" field name: [Optional] Enter custom field name...

"E-mail" field name: [Optional] Enter custom field name...

"BugID" field name: [Optional] Enter custom field name...

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to additional information

□ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

Mantis send method options

1. "**URL**" (`.SendMantisURL`) option specifies target URL for your Mantis installation. Do not add here "http://" or ":80" parts here. Specify only domain name (or IP) and path. Example: `www.example.com/mantis/`

Warning: be sure to setup adequate maximum upload file limits in your web-server/Mantis configuration. Otherwise sending may fail on large bug reports.

2. "**Port**" (`.SendMantisPort`) option specifies HTTP port on web server. It's 80 by default. Other common value is 8080. For SSL/TLS it's usually 443. Port will be set automatically to 80/443 by default.

3. "**SSL / TLS**" (`.SendMantisSSL`) option enabled secure mode (HTTPS protocol). Port will be set automatically to 80/443 by default. Don't forget to adjust port number, if you are using alternative port number.

4. "**Login**" (`.SendMantisLogin`) and "**Password**" (`.SendMantisPassword`) options specify your account on Mantis server. This account will be used to submit bug reports. Do **not** specify e-mail or full user name as login, use only user name. You can use API token instead of password - however, you still have to supply a valid login (user name).

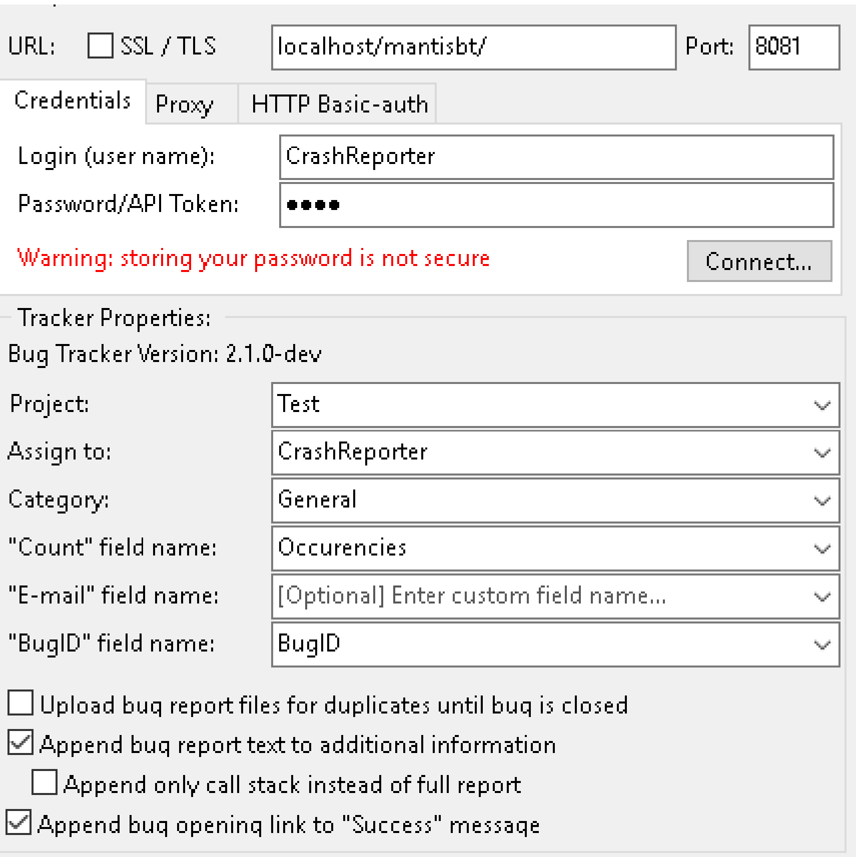
Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use Mantis admin account here. Create a new special account for bug reporting via this method. Limit its rights to submitting only.

Note: it is a good idea to disable e-mail notifications for this account. It is a good idea to make this account "protected".

5. "**BasicAuth**" (`.SendMantisBasicAuthLogin`, `.SendMantisBasicAuthPassword`) options specify account details for Basic-Auth type authentication. Usually these fields are blank. Fill them, if you use Basic-Auth on your web server.

6. "**Proxy**" (.SendMantisProxyHost, .SendMantisProxyPort, .SendMantisProxyLogin, .SendMantisProxyPassword) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

7. "**Connect**" button will try to connect to your Mantis server using the specified URL/port/credentials. If you made mistake in your configuration - an error message will be displayed. In case of success - field below will be populated from configuration of your bug tracker.



URL: □ SSL / TLS localhost/mantisbt/ Port: 8081

Login (user name): CrashReporter

Password/API Token: ••••

Warning: storing your password is not secure

Tracker Properties:

Bug Tracker Version: 2.1.0-dev

Project: Test

Assign to: CrashReporter

Category: General

"Count" field name: Occurencies

"E-mail" field name: [Optional] Enter custom field name...

"BugID" field name: BugID

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to additional information

□ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

8. "**Project**" (.SendMantisProject) options specifies project name to store bug reports. It's mandatory.

9. "**Assign to**" (.SendMantisOwner) option specifies owner account name. If this option is empty, all submitted bug reports will be unassigned (and their state will be "new"). If you enter here any account name (it could be the same as "Login" option) - all submitted bug reports will be assigned to this account (and their state will be "assigned").

Note: this option is ignored, if submitted bug report already exists (submitting a known issue).

10. "**Category**" (.SendMantisCategory) option specifies category for submitted reports. It can be optional or mandatory - it depends on your Mantis version and project configuration.

11. "**"Count" field name**" (.SendMantisCountFieldName) option specifies name of custom field, which EurekaLog will use for bug report counting. By default, Mantis doesn't have any "occurrences" or "count" fields, so you can't know how many times bug has occurred. To workaround this problem, you can create a custom field in Mantis configuration, which you will use for this purpose. You can enter name of this field here - and EurekaLog will use it to count bugs.

Note: we highly recommend to create and use this field.

12. "**"E-mail" field name**" (`.SendMantisEMailFieldName`) option specifies name of custom field, which EurekaLog will use to store e-mail of user who sent (original) report. By default, Mantis doesn't have such field. You can either create a custom field in Mantis configuration, which you will use for this purpose; or you can simply extract e-mail from bug report (file attach).

13. "**"BugID" field name**" (`.SendMantisBugIDFieldName`) option specifies name of custom field, which EurekaLog will use to store BugID. By default, Mantis doesn't have such field. You can create a custom field in Mantis configuration, which you will use for this purpose. When field is specified - it will be used by EurekaLog to search/merge reports. Otherwise title (summary) is used for merging.

Note: we highly recommend to create and use this field.

12. "**Upload bug report files for duplicates until bug is closed**" (`.SendMantisUploadFilesForDups`) option allows you to collect all bug reports. If this option is unchecked (default): only first bug report is uploaded and stored. All other bug reports for the same problem (identified by BugID) will be discarded. Only "count" field will be increased (if it was configured). If this option is checked: bug reports for the same problem will be uploaded to issue.

Notes:
- if you check this option be sure to name bug report files in unique way to avoid file names duplicates. Also, be sure to have a lot of hard disk space to store all bug reports.
- bug reports will not be uploaded, once the issue is closed or resolved.
- you can also stop collecting files by changing status to "confirmed" or "acknowledged".

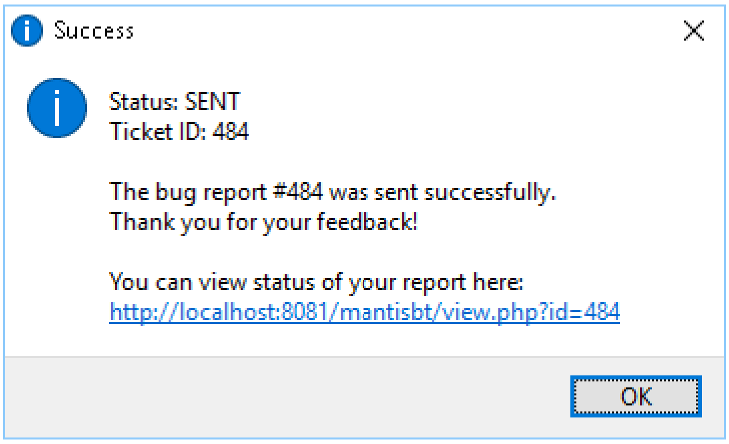
13. "**Append bug report text to additional information**" (`.SendMantisAppendText`) option allows you to insert bug report's text into "Additional information" field. It's convenient, if you need to peek bug report without downloading bug report file. You can turn this option off, if you don't need this behaviour.

Note: checking this option will not disable bug report file upload. File will still be attached.

14. "**Append only call stack instead of full report**" (`.SendMantisUseCallStackAsBugReport`) option alters previous option. Disabled: full bug report text will be added (e.g. general section, call stack, modules, processes, CPU/assembler, etc.); Enabled: only call stack will be added (you will still be able to view full bug report by downloading file attach).

15. "**Append bug opening link to "Success" message**" (`.SendMantisAllowLinks`) option will add a link to view bug report on Mantis to message dialog after successful send. So end user (client) will be able to view status of the report on your bug tracker (login is required).

This option has no effect if successful message dialog is disabled. Turn this option on for public bug trackers. Turn this option off for private bug trackers.

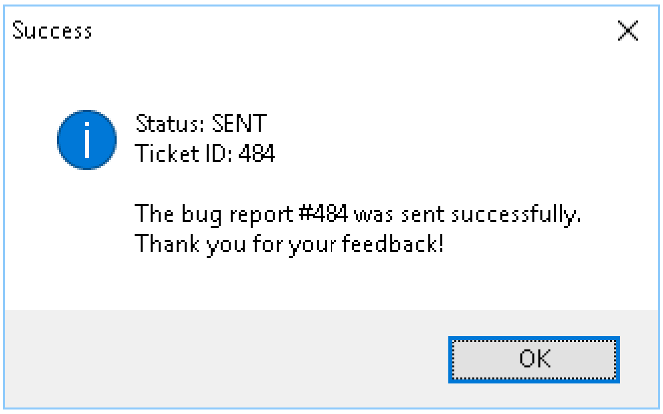


Status: SENT
Ticket ID: 484

The bug report #484 was sent successfully.
Thank you for your feedback!

You can view status of your report here:
http://localhost:8081/mantisbt/view.php?id=484

"Append bug opening link to "Success" message" option is enabled



Success

Status: SENT
Ticket ID: 484

The bug report #484 was sent successfully.
Thank you for your feedback!

"Append bug opening link to "Success" message" option is disabled

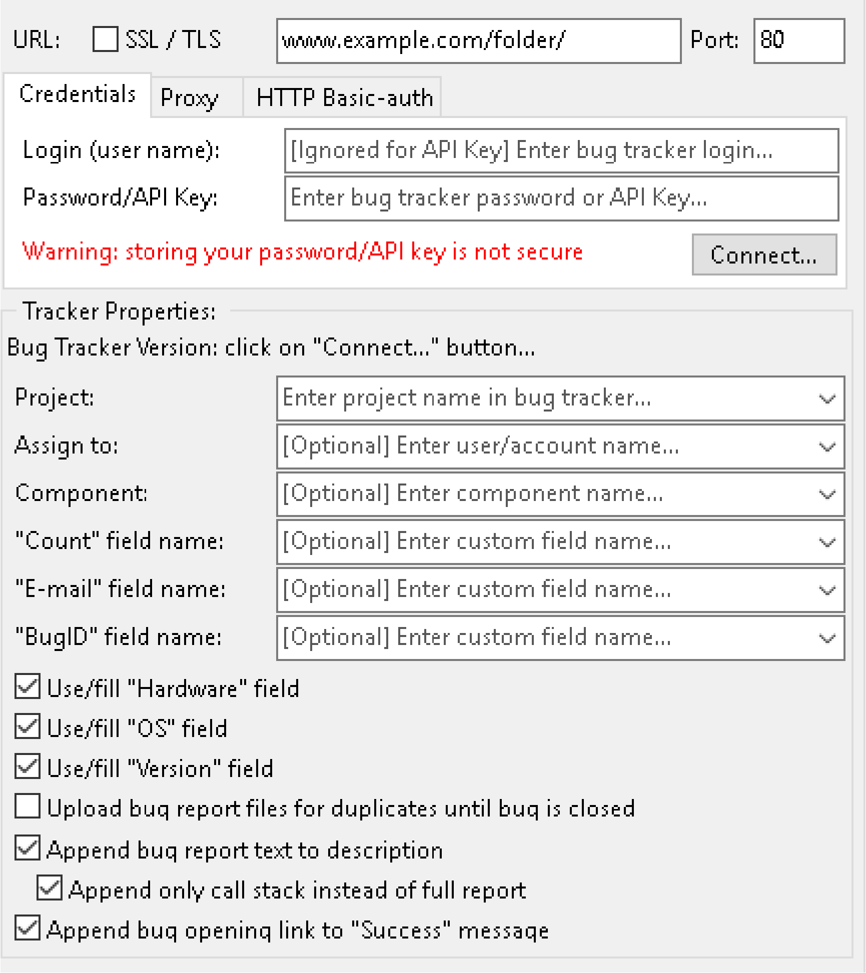
Note: active hyper-link will work on Windows Vista or later. It will be displayed as plain text on Windows XP and earlier.

See also:

- Mantis send method 406 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158
- Managing bug reports in Mantis 105
- Mantis setup 119

10.3.5.11 BugZilla

This is setup options for BugZilla send method 407 (wsmBugZilla). They are located at Sending tab 302.



URL: □ SSL / TLS www.example.com/folder/ Port: 80

Login (user name): [Ignored for API Key] Enter bug tracker login...

Password/API Key: Enter bug tracker password or API Key...

Warning: storing your password/API key is not secure

Tracker Properties:

Bug Tracker Version: click on "Connect..." button...

Project: Enter project name in bug tracker...

Assign to: [Optional] Enter user/account name...

Component: [Optional] Enter component name...

"Count" field name: [Optional] Enter custom field name...

"E-mail" field name: [Optional] Enter custom field name...

"BugID" field name: [Optional] Enter custom field name...

☑ Use/fill "Hardware" field

☑ Use/fill "OS" field

☑ Use/fill "Version" field

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

BugZilla send method options

1. "**URL**" (.SendBugZillaURL) option specifies target URL for your BugZilla installation. Do not add here "http://" or ":80" parts here. Specify only domain name (or IP) and path. Example: `www.example.com/bugzilla/`

Warning: be sure to setup adequate maximum upload file limits in your web-server/BugZilla configuration. Otherwise sending may fail on large bug reports.

2. "**Port**" (.SendBugZillaPort) option specifies HTTP port on web server. It's 80 by default. Other common value is 8080. For SSL/TLS it's usually 443. Port will be set automatically to 80/443 by default.

3. "**SSL / TLS**" (.SendBugZillaSSL) option enabled secure mode (HTTPS protocol). Port will be set automatically to 80/443 by default. Don't forget to adjust port number, if you are using alternative port number.

4. "**Login**" (.SendBugZillaLogin) and "**Password**" (.SendBugZillaPassword) options specify your account on BugZilla server. This account will be used to submit bug reports. You can use API token instead of password - leave login field blank in this case.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use BugZilla admin account here. Create a new special account for bug reporting via this method. Limit its rights to submitting only.

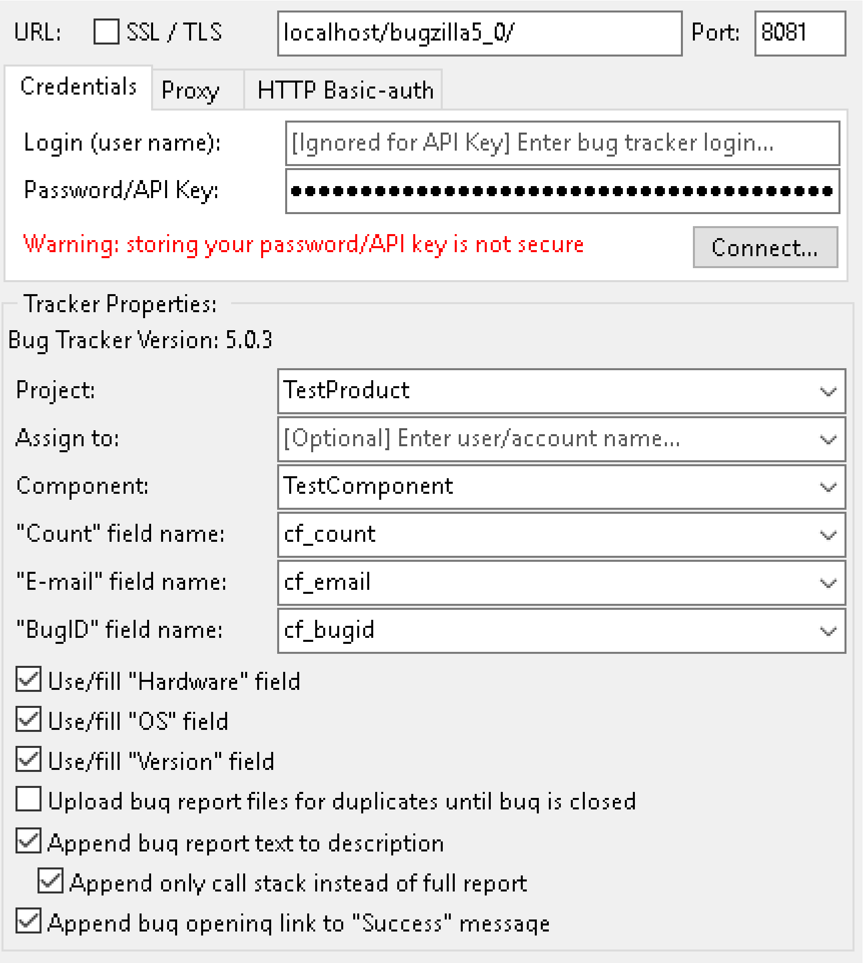
Note: it is a good idea to disable e-mail notifications for this account.

5. "**BasicAuth**" (.SendBugZillaBasicAuthLogin, .SendBugZillaBasicAuthPassword) options

specify account details for Basic-Auth type authentication. Usually these fields are blank. Fill them, if you use Basic-Auth on your web server.

6. "**Proxy**" (.SendBugZillaProxyHost, .SendBugZillaProxyPort, .SendBugZillaProxyLogin, .SendBugZillaProxyPassword) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

7. "**Connect**" button will try to connect to your BugZilla server using the specified URL/port/credentials. If you made mistake in your configuration - an error message will be displayed. In case of success - field below will be populated from configuration of your bug tracker.



URL: □ SSL / TLS localhost/bugzilla5_0/ Port: 8081

Credentials | Proxy | HTTP Basic-auth

Login (user name): [Ignored for API Key] Enter bug tracker login...

Password/API Key: ●●

Warning: storing your password/API key is not secure

Tracker Properties:

Bug Tracker Version: 5.0.3

Project: TestProduct

Assign to: [Optional] Enter user/account name...

Component: TestComponent

"Count" field name: cf_count

"E-mail" field name: cf_email

"BugID" field name: cf_bugid

☑ Use/fill "Hardware" field

☑ Use/fill "OS" field

☑ Use/fill "Version" field

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

8. "**Project**" (.SendBugZillaProject) options specifies project name to store bug reports. It's mandatory.

9. "**Assign to**" (.SendBugZillaOwner) option specifies owner account name. If this option is empty, all submitted bug reports will be assigned to default account (configured in BugZilla options). If you enter here any account name - all submitted bug reports will be assigned to this account.

Note: this option is ignored, if submitted bug report already exists (submitting a known issue).

10. "**Component**" (.SendBugZillaComponent) option specifies category for submitted reports. It can be optional or mandatory - it depends on your BugZilla version and project configuration.

11. "**"Count" field name**" (.SendBugZillaCountFieldName) option specifies name of custom field, which EurekaLog will use for bug report counting. By default, BugZilla doesn't have

any "occurrences" or "count" fields, so you can't know how many times bug has occurred. To workaround this problem, you can create a custom field in BugZilla configuration, which you will use for this purpose. You can enter name of this field here - and EurekaLog will use it to count bugs.

Note: we highly recommend to create and use this field.

12. "**"E-mail" field name**" (`.SendMantisEMailFieldName`) option specifies name of custom field, which EurekaLog will use to store e-mail of user who sent (original) report. By default, BugZilla doesn't have such field. You can either create a custom field in BugZilla configuration, which you will use for this purpose; or you can simply extract e-mail from bug report (file attach).

13. "**"BugID" field name**" (`.SendMantisBugIDFieldName`) option specifies name of custom field, which EurekaLog will use to store BugID. By default, BugZilla doesn't have such field. You can create a custom field in BugZilla configuration, which you will use for this purpose. It is purely optional, as EurekaLog will use internal alias for merging.

14. "**Use hardware field**" (`.SendBugZillaUseHardware`) option instructs EurekaLog to use "Hardware" field in BugZilla. Disable this option, if you've customized this field and/or use it for other purposes. You can also fill this field manually instead of automatic generation by EurekaLog.

15. "**Use OS field**" (`.SendBugZillaUseOS`) option instructs EurekaLog to use "OS" field in BugZilla. Disable this option, if you've customized this field and/or use it for other purposes. You can also fill this field manually instead of automatic generation by EurekaLog.

16. "**Use version field**" (`.SendBugZillaUseVersion`) option instructs EurekaLog to use "Version" field in BugZilla. Disable this option, if you've customized this field and/or use it for other purposes. You can also fill this field manually instead of automatic generation by EurekaLog.

Note: you must enter valid versions into BugZilla's configuration. You also need to store version information in your executables to fill this field.

17. "**Upload bug report files for duplicates until bug is closed**" (`.SendBugZillaUploadFilesForDups`) option allows you to collect all bug reports. If this option is unchecked (default): only first bug report is uploaded and stored. All other bug reports for the same problem (identified by BugID) will be discarded. Only "count" field will be increased (if it was configured). If this option is checked: bug reports for the same problem will be uploaded to issue.

Notes:
- if you check this option be sure to name bug report files in unique way to avoid file names duplicates. Also, be sure to have a lot of hard disk space to store all bug reports.
- bug reports will not be uploaded, once the issue is closed or resolved.
- you can also stop collecting files by changing status to "VERIFIED" or "IN_PROGRESS".

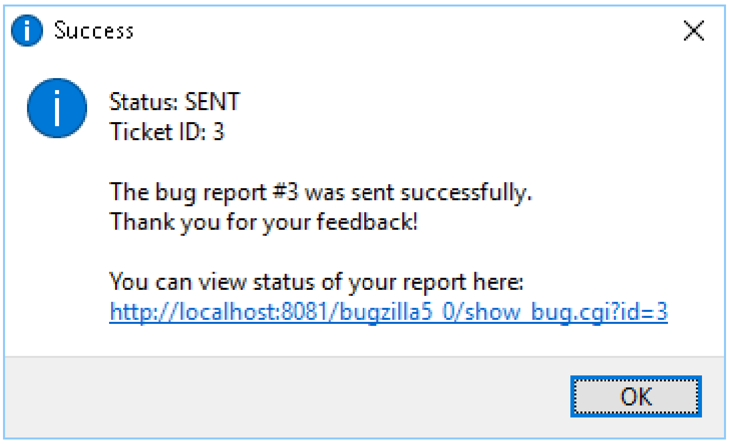
18. "**Append bug report text to additional information**" (`.SendBugZillaAppendText`) option allows you to insert bug report's text into "Description" field. It's convenient, if you need to peek bug report without downloading bug report file. You can turn this option off, if you don't need this behaviour.

Note: checking this option will not disable bug report file upload. File will still be attached.

19. "**Append only call stack instead of full report**" (`.SendBugZillaUseCallStackAsBugReport`) option alters previous option. Disabled: full bug report text will be added (e.g. general section, call stack, modules, processes, CPU/ assembler, etc.); Enabled: only call stack will be added (you will still be able to view full bug report by downloading file attach).

20. "**Append bug opening link to "Success" message**" (`.SendBugZillaAllowLinks`) option will add a link to view bug report on BugZilla to message dialog after successful send. So end user (client) will be able to view status of the report on your bug tracker (login is required).

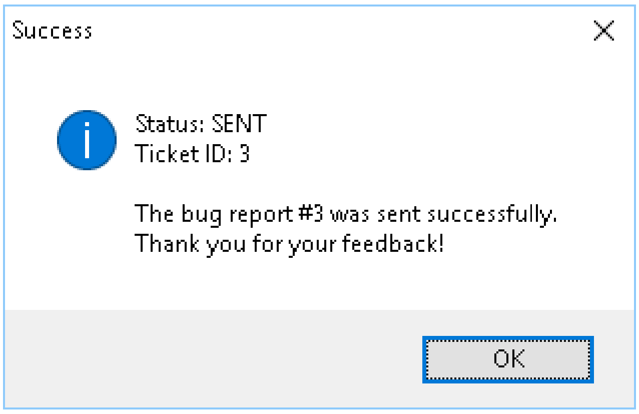
This option has no effect if successful message dialog is disabled. Turn this option on for public bug trackers. Turn this option off for private bug trackers.


Status: SENT
Ticket ID: 3

The bug report #3 was sent successfully.
Thank you for your feedback!

You can view status of your report here:
http://localhost:8081/bugzilla5_0/show_bug.cgi?id=3

"Append bug opening link to "Success" message" option is enabled


Status: SENT
Ticket ID: 3

The bug report #3 was sent successfully.
Thank you for your feedback!

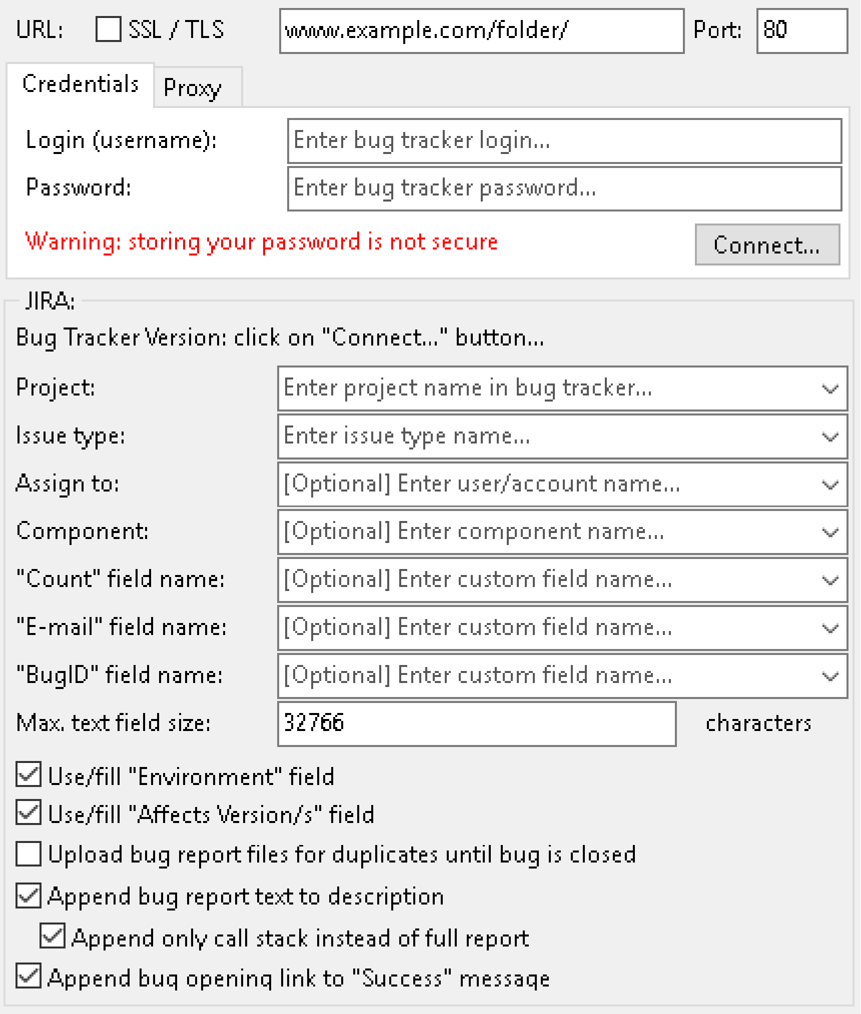
"Append bug opening link to "Success" message" option is disabled

Note: active hyper-link will work on Windows Vista or later. It will be displayed as plain text on Windows XP and earlier.

See also:
- BugZilla send method 407 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158
- Managing bug reports in BugZilla 105
- BugZilla setup 134

10.3.5.12 JIRA

This is setup options for JIRA send method 408 (`wsmJIRA`). They are located at Sending tab 302 .



URL: □ SSL / TLS www.example.com/folder/ Port: 80

Login (username): Enter bug tracker login...

Password: Enter bug tracker password...

Warning: storing your password is not secure

JIRA:

Bug Tracker Version: click on "Connect..." button...

Project: Enter project name in bug tracker...

Issue type: Enter issue type name...

Assign to: [Optional] Enter user/account name...

Component: [Optional] Enter component name...

"Count" field name: [Optional] Enter custom field name...

"E-mail" field name: [Optional] Enter custom field name...

"BugID" field name: [Optional] Enter custom field name...

Max. text field size: 32766 characters

☑ Use/fill "Environment" field

☑ Use/fill "Affects Version/s" field

□ Upload bug report files for duplicates until bug is closed

☑ Append bug report text to description

☑ Append only call stack instead of full report

☑ Append bug opening link to "Success" message

JIRA send method options

1. "**URL**" (.SendJIRAURL) option specifies target URL for your JIRA installation. Do not add here "http://" or ":80" parts here. Specify only domain name (or IP) and path. Example: `www.example.com/jira/`

Warning: be sure to setup adequate maximum upload file limits in your web-server/JIRA configuration. Otherwise sending may fail on large bug reports.

2. "**Port**" (.SendJIRAPort) option specifies HTTP port on web server. It's 80 by default. Other common value is 8080. For SSL/TLS it's usually 443. Port will be set automatically to 80/443 by default.

3. "**SSL / TLS**" (.SendJIRASSL) option enabled secure mode (HTTPS protocol). Port will be set automatically to 80/443 by default. Don't forget to adjust port number, if you are using alternative port number.

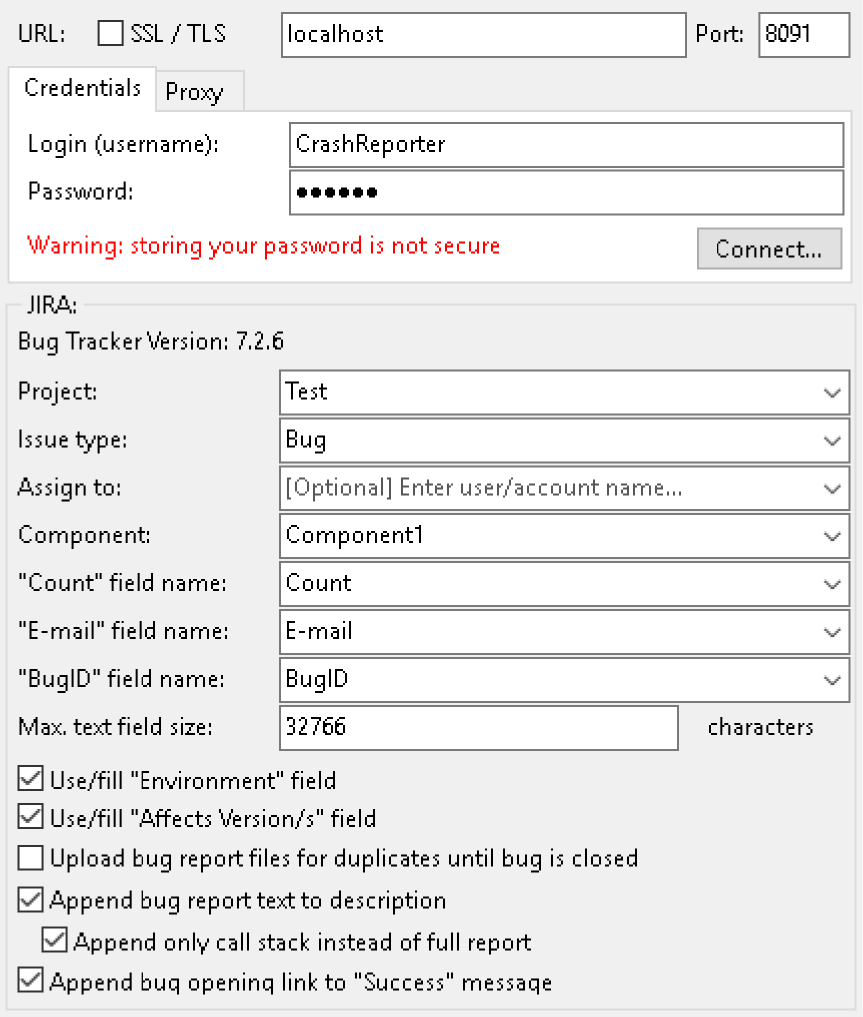
4. "**Login**" (.SendJIRALogin) and "**Password**" (.SendJIRAPassword) options specify your account on JIRA server. This account will be used to submit bug reports. Do **not** specify e-mail or full user name as login, use only username. You can find your username in your JIRA profile.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use JIRA admin account here. Create a new special account for bug reporting via this method. Limit its rights to submitting only.

Note: it is a good idea to disable e-mail notifications for this account.

5. "**Proxy**" (.SendJIRAProxyHost, .SendJIRAProxyPort, .SendJIRAProxyLogin, .SendJIRAProxyPassword) options specify proxy details. You can leave them blank to use system-provided settings. Or you can fill these values to set custom proxy.

6. "**Connect**" button will try to connect to your JIRA server using the specified URL/port/ credentials. If you made mistake in your configuration - an error message will be displayed. In case of success - field below will be populated from configuration of your bug tracker.



URL: □ SSL / TLS localhost Port: 8091

Login (username): CrashReporter

Password: ●●●●●●

Warning: storing your password is not secure

JIRA:

Bug Tracker Version: 7.2.6

Project:	Test
Issue type:	Bug
Assign to:	[Optional] Enter user/account name...
Component:	Component1
"Count" field name:	Count
"E-mail" field name:	E-mail
"BugID" field name:	BugID
Max. text field size:	32766 characters

☑ Use/fill "Environment" field
☑ Use/fill "Affects Version/s" field
□ Upload bug report files for duplicates until bug is closed
☑ Append bug report text to description
☑ Append only call stack instead of full report
☑ Append bug opening link to "Success" message

7. "**Project**" (.SendJIRAProject) options specifies project name to store bug reports. It's mandatory. Project name is case-insensitive, do not specify project key as name.

8. "**Issue type**" (.SendJIRAIssueType) option specifies issue type to create. Mandatory. Default is "Bug".

9. "**Assign to**" (.SendJIRAOwner) option specifies owner account name. If this option is empty, all submitted bug reports will be assigned to default account (configured in JIRA options). If you enter here any account name - all submitted bug reports will be assigned to this account. Do not specify full user name as login, use only username.

Note: this option is ignored, if submitted bug report already exists (submitting known issue).

10. "**Component**" (.SendJIRAComponent) option specifies component for submitted reports. It's optional.

11. "**"Count" field name**" (`.SendJIRACountFieldName`) option specifies name of custom field, which EurekaLog will use for bug report counting. By default, JIRA doesn't have any "occurrences" or "count" fields, so you can't know how many times bug has occurred. To workaround this problem, you can create a custom field in JIRA configuration, which you will use for this purpose. You can enter name of this field here - and EurekaLog will use it to count bugs.

12. "**"E-mail" field name**" (`.SendJIRAEMailFieldName`) option specifies name of custom field, which EurekaLog will use to store e-mail of user who sent (original) report. By default, JIRA doesn't have such field. You can either create a custom field in JIRA configuration, which you will use for this purpose; or you can simply extract e-mail from bug report (file attach).

13. "**"BugID" field name**" (`.SendJIRABugIDFieldName`) option specifies name of custom field, which EurekaLog will use to store BugID. By default, JIRA doesn't have such field. You can create a custom field in JIRA configuration, which you will use for this purpose. When field is specified - it will be used by EurekaLog to search/merge reports. Otherwise title (summary) is used for merging.

Note: we highly recommend to create and use this field.

14. "**Upload bug report files for duplicates until bug is closed**" (`.SendJIRAUploadFilesForDups`) option allows you to collect all bug reports. If this option is unchecked (default): only first bug report is uploaded and stored. All other bug reports for the same problem (identified by BugID) will be discarded. Only "count" field will be increased (if it was configured). If this option is checked: bug reports for the same problem will be uploaded to issue.

Notes:
- if you check this option be sure to name bug report files in unique way to avoid file names duplicates. Also, be sure to have a lot of hard disk space to store all bug reports.
- bug reports will not be uploaded, once the issue is closed or resolved.
- you can also stop collecting files by changing status to "In progress".

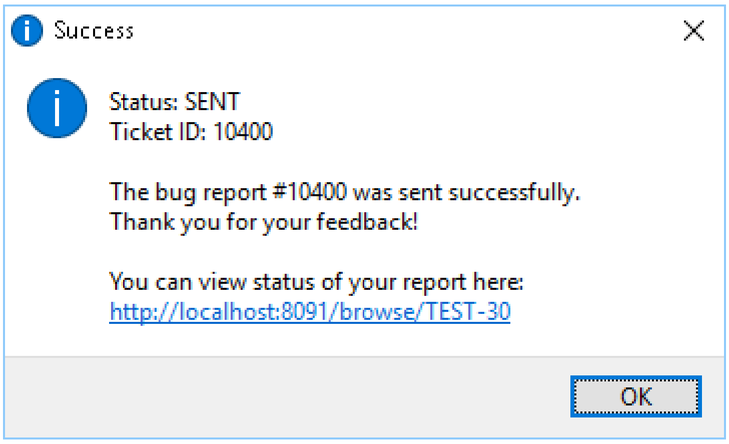
15. "**Append bug report text to additional information**" (`.SendJIRAAppendText`) option allows you to insert bug report's text into "Description" field. It's convenient, if you need to peek bug report without downloading bug report file. You can turn this option off, if you don't need this behaviour.

Note: checking this option will not disable bug report file upload. File will still be attached.

15. "**Append only call stack instead of full report**" (`.SendJIRAUseCallStackAsBugReport`) option alters previous option. Disabled: full bug report text will be added (e.g. general section, call stack, modules, processes, CPU/assembler, etc.); Enabled: only call stack will be added (you will still be able to view full bug report by downloading file attach).

16. "**Append bug opening link to "Success" message**" (`.SendJIRAAllowLinks`) option will add a link to view bug report on BugZilla to message dialog after successful send. So end user (client) will be able to view status of the report on your bug tracker (login is required).

This option has no effect if successful message dialog is disabled. Turn this option on for public bug trackers. Turn this option off for private bug trackers.



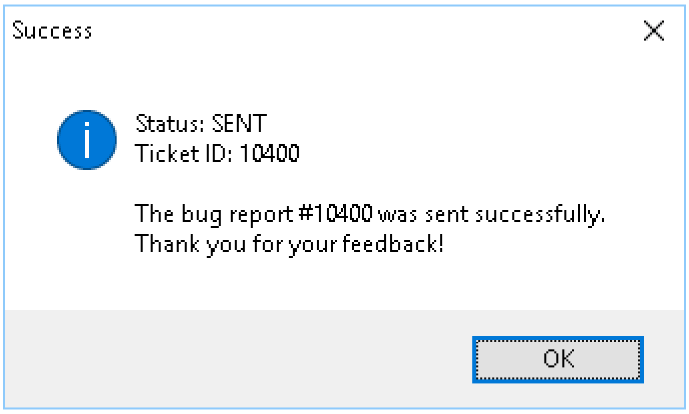
Success

Status: SENT
Ticket ID: 10400

The bug report #10400 was sent successfully.
Thank you for your feedback!

You can view status of your report here:
http://localhost:8091/browse/TEST-30

"Append bug opening link to "Success" message" option is enabled



Success

Status: SENT
Ticket ID: 10400

The bug report #10400 was sent successfully.
Thank you for your feedback!

"Append bug opening link to "Success" message" option is disabled

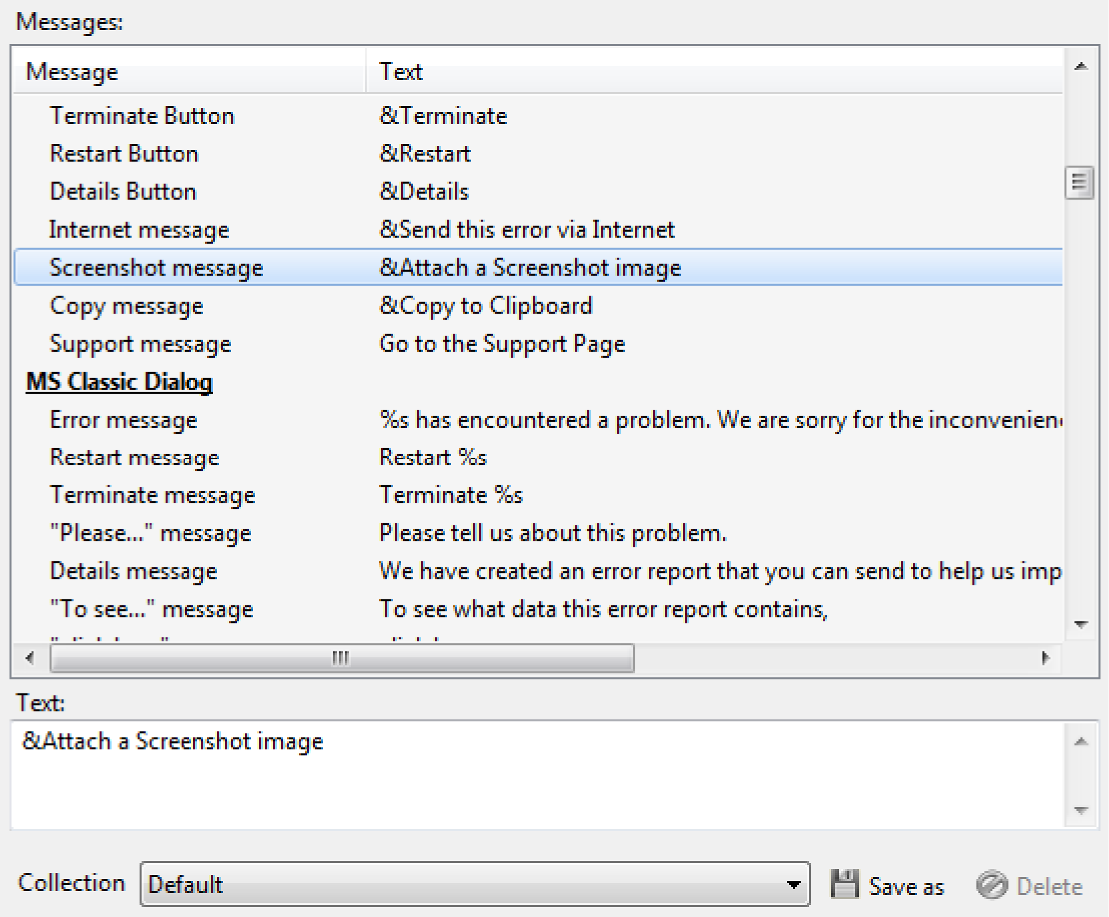
Note: active hyper-link will work on Windows Vista or later. It will be displayed as plain text on Windows XP and earlier.

See also:

- JIRA send method 408 for general description of this send method
- General send options 304
- Configuring send method 53
- Security Considerations 158
- Managing bug reports in JIRA 105
- JIRA setup 143

10.3.6 Localization page

This is "Localization" page in EurekaLog project's options 225.



Messages:

Message	Text
Terminate Button	&Terminate
Restart Button	&Restart
Details Button	&Details
Internet message	&Send this error via Internet
Screenshot message	&Attach a Screenshot image
Copy message	&Copy to Clipboard
Support message	Go to the Support Page
MS Classic Dialog	
Error message	%s has encountered a problem. We are sorry for the inconvenien
Restart message	Restart %s
Terminate message	Terminate %s
"Please..." message	Please tell us about this problem.
Details message	We have created an error report that you can send to help us imp
"To see..." message	To see what data this error report contains,

Text:

&Attach a Screenshot image

Collection: Default

Localization options

This page allows you to translate EurekaLog to another language.

1. "**Messages**". The list of messages can be used to select individual message text.

2. "**Text**". Once line is selected - you can edit text in memo control.

3. "**Collection**" option can be used to save localized texts into a file, which can be used later or copied to another machine.

To load message texts from existing file - select it from "Collection" combobox. "Default" position will revert all texts to default (English). "Custom" position is selected automatically when you edit existing collection:

- When "Collection" option shows "Custom" - all localized texts will be saved to your configuration.
- When "Collection" option shows any other value - localized texts are not saved to your project configuration. Instead a collection name will be saved. Please note that your machine must have corresponding .etf file. Otherwise collection will be reset to "Default".

Click on "Save as" button to save collection of message texts into a new or existing .etf file. Existing collection file will be overwritten. Simply enter a name: do not specify file extension or any special characters - such as:

```
    \ / : * ? " < > | . , ;
```

Alternatively, you can also specify a full file name (with .etf extension) to save collection to a different folder. You can use %_IDESrc% environment variable 413 to specify paths relative to your project's folder. For example:

```
    %_IDESrc%Localizations\English.etf
```

This will save collection into "Localizations" sub-folder of your project.

Important Note: any already saved localized texts in your project's configuration are ignored - unless "Collection" option is set to "Custom" or collection's file could not be found.

Use these options to translate EurekaLog's messages to another language. You can use these options if you do not use any localization software. However, if you're using some sort of localization solution - then you should switch this to "Default" and use your localization software (such as ITE, TsiLang, dxGetText, etc.).

All message texts in EurekaLog come through this path:

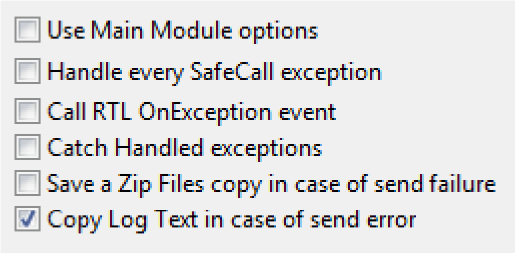
resourcestring -> OnTranslate -> Options -> UI

1. Any message text starts as string in resources section of executable. You can change these lines with translation software tool, which is able to work with resource (such as ITE, TsiLang, etc.).
2. Then each text is passed to `OnTranslate` event (`EStrConsts` unit). You can assign your own handler to translate texts. Use this event for translation software which supports GetText-like function (such as dxGetText).
3. Then each text is passed to module options, where it can be overridden by localization options (set on "Localization" tab in EurekaLog project options). You can also alter it at run-time via indexed `.CustomizedTexts` property.
4. Final result is displayed in UI.

Note: "Right-To-Left" value under "Dialogs (common)" section defines Left-To-Right or Right-To-Left layout for all EurekaLog run-time dialogs. Value of 0 indicate left-to-right layout (default), value of 1 indicate right-to-left layout used in some middle eastern languages. This option can also be altered at design-time via Dialogs page 267 (for example 271). This option can also be altered at run-time by changing `CurrentEurekaLogOptions.CustomizedTexts[mtRTLDialog]`.

10.3.7 Advanced page

This is "Advanced" page in EurekaLog project's options 225.



□ Use Main Module options
□ Handle every SafeCall exception
□ Call RTL OnException event
□ Catch Handled exceptions
□ Save a Zip Files copy in case of send failure
☑ Copy Log Text in case of send error

Advanced options

This page allows you to set advanced EurekaLog options. There are also some sub-categories:

- Exception filters 343
- Build options 349
- Code injection 352
- Custom/manual options setup 356

1. "**Use main module options**" option will load options from host .exe file (when available). This option have no effect for .exe and BPL files, it's only applicable for DLLs. It is used when you want to share single options between many DLLs. You must compile your .exe and DLL in compatible versions of EurekaLog when you use this option.

It's not recommended to use this option. Consider using DLLs without EurekaLog's code 368 instead. Alternatively you can use external options file 443 to share options between projects.

2. "**Handle every SafeCall exception**" option is used to catch safecall-exceptions with EurekaLog. This option is useful in COM servers, COM applications and other interface-related code.

When this option is off - safecall exceptions will be handled by default processing which usually means losing information about error location.
When this option is on - safecall exceptions will be handled by EurekaLog and then by default processing.

Usually it's a good idea to disable error dialogs and visual feedback for safecall exceptions since these exception will be handled by calling code (which will display error message).

Notes:
- Each safecall exception is considered to be handled exception. Keep that in mind when you setup exception filters or write event handlers.
- This option has no effect if "**Catch handled exceptions**" option is enabled (see below).
- This option requires extended memory manager 250 enabled.
- It's a good idea to include fix for QC report #81725 352 when you use "**Handle every SafeCall exception**" option.
- Internally, "**Handle every SafeCall exception**" option installs hook for ComObj.HandleSafeCallException routine (when low-level hooks are allowed) or scans exception's call stack for _HandleAutoException routine (when low-level hooks are not installed). The later can cause building call stack for all exceptions even with "deferred call stacks" option set.

Alternative for this option is to invoke EurekaLog manually from your SafeCallException handler.

See also:
- Using EurekaLog in COM applications 488

3. "**Call RTL OnException event**" option will invoke default processing for exception after processing exception by EurekaLog. Use this option to get default behaviour (such as error dialogs), but still use EurekaLog features.

Usually it's a good idea to disable error dialogs and visual feedback when enabling this option.

It's not recommended to use this option. It's primary used for backward compatibility with old EurekaLog versions. New code should consider using RTL error dialog 371 instead.

4. "**Catch handled exceptions**" option will enable EurekaLog for all exceptions. By default EurekaLog processes only exceptions which are unhandled (see handled/unhandled terms definitions 40).

It's not recommended to use this option. That's because "handled" for exception means that this exception is expected and it was handled by code. Therefore, it's better to setup proper exception handling in your code. This option is used primary as last resort measure to work with "bad" code (the code which hides unhandled exceptions).

Be sure to setup proper exception filtering when you enable this option. Often it's a good idea to disable error dialogs and visual feedback for handled exceptions.

Note: you should use "**Handle every SafeCall exception**" option for safecall-exceptions instead of "**Catch handled exceptions**" option.
Note: this option requires extended memory manager 250 enabled.

5. "**Save a ZIP file copy in case of send failure**" (`.boSaveCompressedCopyInCaseOfError`) option will save bug report file copy in "My Documents" folder if sending fail. Use this option to allow end user to send bug report manually to you if send fails (for some reason). This option has no effect if no sending method was set up.

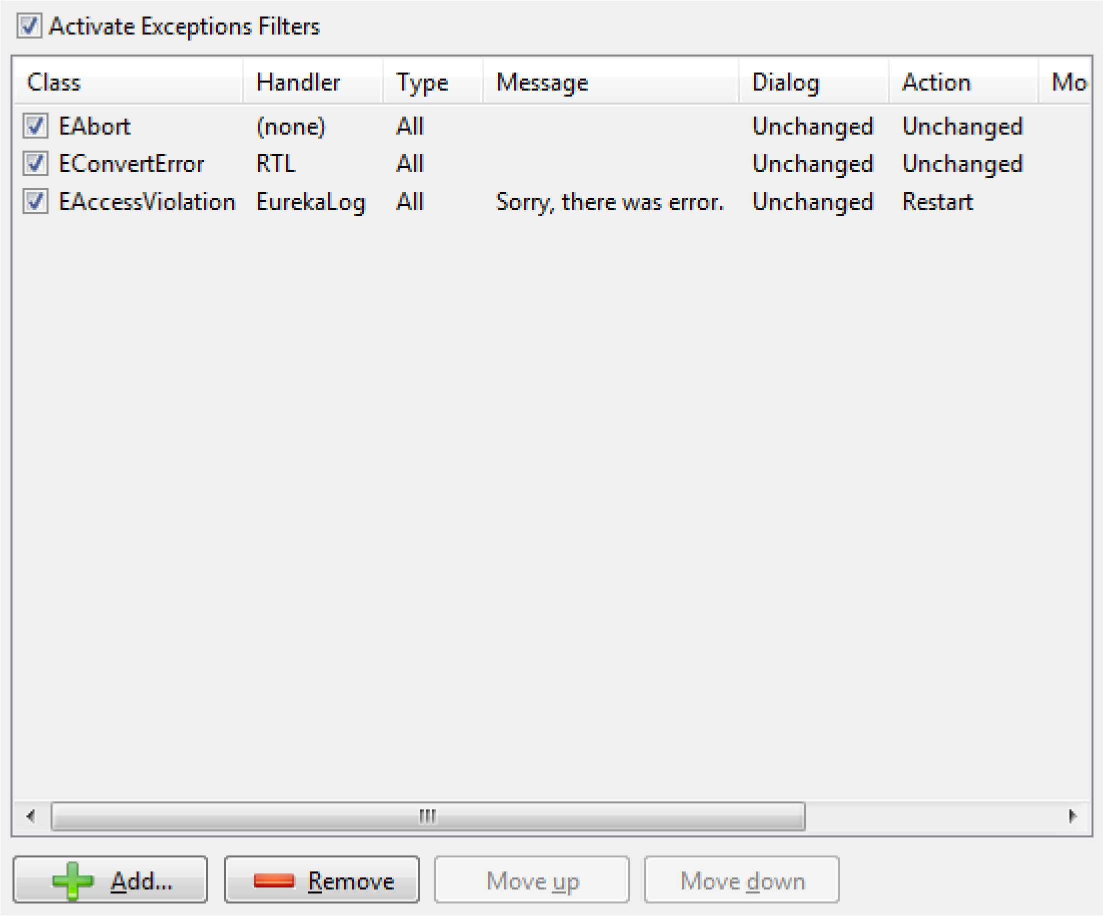
6. "**Copy log text in case of send error**" (`.boCopyLogInCaseOfError`) option will copy bug report text into Windows clipboard if sending fail.

See also:
- Exception filters 343

- Build options 349
- Code injection 352
- External tools options 259
- Custom/manual options setup 356
- Using EurekaLog in COM applications 488

10.3.7.1 Exceptions filters page

This is "Advanced/Exception filters" page in EurekaLog project's options 225.



☑ Activate Exceptions Filters

Class	Handler	Type	Message	Dialog	Action	Mo
☑ EAbort	(none)	All		Unchanged	Unchanged	
☑ EConvertError	RTL	All		Unchanged	Unchanged	
☑ EAccessViolation	EurekaLog	All	Sorry, there was error.	Unchanged	Restart	

Exception filters options

This page allows you to set up exception filters. Exception filter is a filter which can alter EurekaLog's behavior based on some properties of exception. It is a easy way to customize EurekaLog on per-exception basis without writing code 185.

1. "**Activate Exceptions Filters**" (`.ActivateFilters`) option enables or disables exception filters globally. Exceptions filter below will have no effect when this option is disabled.

2. "**Exception Filters**" (`.ExceptionsFilters`) option defines one or more exception filter.

Usually exceptions are identified by exception's class name. You can also identify exception by source location. And you can identify exception by its type (handled or unhandled). Once exception is identified - you can change handler for it (none, RTL or EurekaLog), dialog class (to any of existing dialog classes), override error message or set action (restart or terminate). Exception filters are applied before processing exception. Filters are applied in order from top to bottom. First matched filter is applied.

When exceptions filters are not enough - you have to write code (event handlers).

- Click on "Add" button to add new filter 344.

- Select existing filter and click on "Remove" button to delete selected filter.
- You can double-click on existing filter to edit (modify) selected filter 344.
- You can use "Move up"/"Move down" buttons, Ctrl + Up/Ctrl + Down or simple mouse dragging to re-order filters.

Note: sometimes it's more easier to use custom attributes 190 instead of exception filters.

Important note: ensure that features specified in exception filters will be available at run-time. For example, if your application uses MS Classic-styled exception dialog by default and you want to switch to EurekaLog-styled dialog for some exceptions by using exception filters - then be sure to include code for EurekaLog dialog into your application 354.

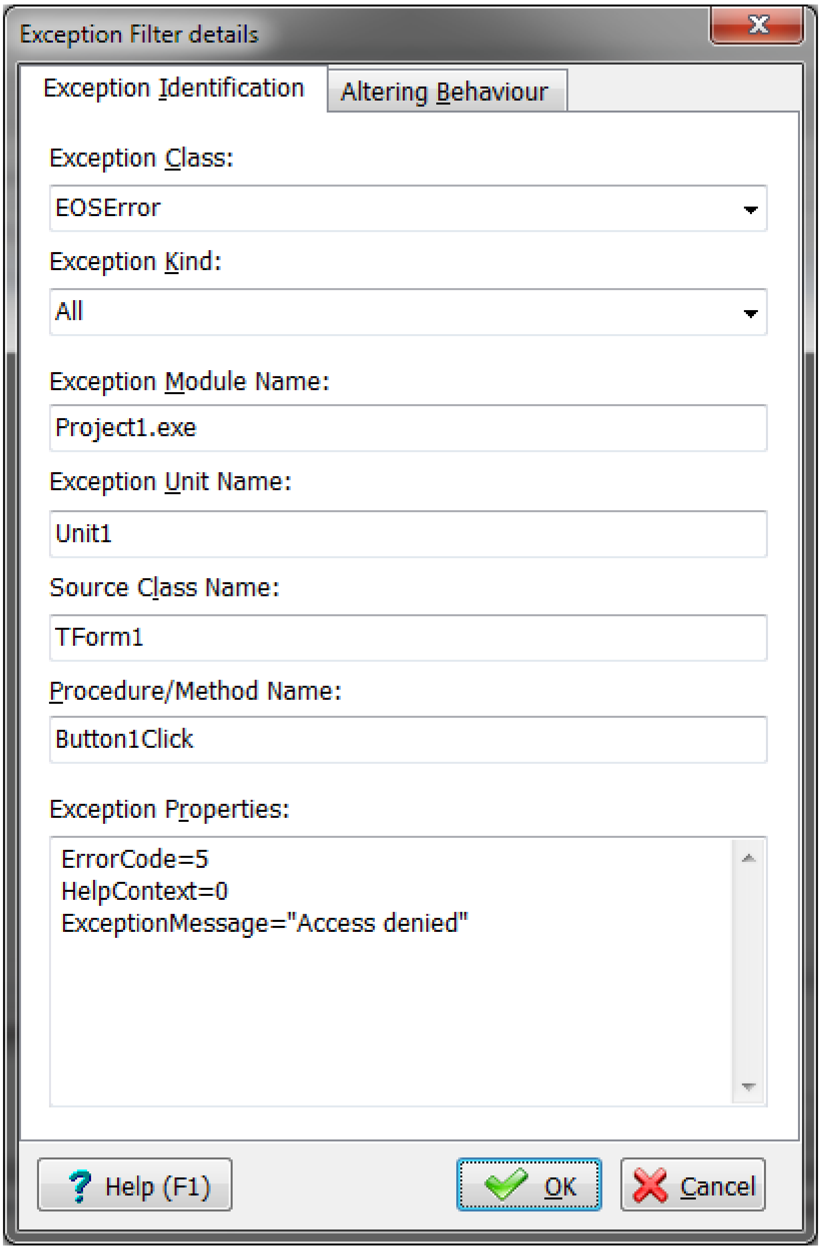
See also:

- Editing exception filter 344
- Using exception filters for customizing EurekaLog 185
- Using custom attributes 190

10.3.7.1.1 Editing exception filter

This is exception filter dialog which is used to add or edit exception filters 343.

Dialog consists of two pages:



Exception Filter details

Exception Class: EOSError

Exception Kind: All

Exception Module Name: Project1.exe

Exception Unit Name: Unit1

Source Class Name: TForm1

Procedure/Method Name: Button1Click

Exception Properties:

```
ErrorCode=5
HelpContext=0
ExceptionMessage="Access denied"
```

Exception Identification Page

Exception Identification Page contains filter options which are used to find matched exception. When exception occurs:

1. EurekaLog walks through all available exception filters.
2. Each filter is compared against current exception.
3. If current exception matches "identification" properties of exception filter - search is over and filter is applied to exception.

Available options are:

1. "**Exception Class**" option specifies exception class name. This is **mandatory** option. Filter will be applied only for exceptions of the specified class. You can pick predefined classes from combo-box (such as `EAbort` or `EAccessViolation`) or enter your own classes (such as `EMyException`). If you want to catch all exceptions - specify "Exception" class.

Important note: your exception classes must be real classes. You can not use aliases. For example:

```
type
  EMyException1 = class(Exception);
  EMyException2 = Exception;
```

You can specify 'EMyException1' as exception class name. However, 'EMyException2' will not work. Because there is no such class. You should use 'Exception' instead.

2. "**Exception Kind**" option specifies whenever this filter will be applicable to unhandled exceptions, handled exceptions, SafeCall exceptions or all of them. This is **mandatory** option.

SafeCall exceptions are considered to be handled exceptions. "Handled" value will catch any kind of handled exception - regardless of it being SafeCall exception. "SafeCall" value will catch only SafeCall exceptions, but no other handled exceptions.

Tip: Filter is applicable for all exceptions by default. Different processing for handled and unhandled exceptions may be confusing, so it's recommended to avoid it when possible.

Note: this option is ignored if your application do not process handled or SafeCall exceptions.

3. "**Exception Module Name**" option specifies name of file for executable module. Such as 'Project1.exe' or 'BugDLL.dll'. This option is optional. Filter will be applied only to exceptions raised from the specified module.

Notes:
- You must specify file name only without any file path.
- Module name comparison follows case sensitivity of host OS.
- This option uses ExceptionAddress property to determinate module name.
- This option does not require debug information to work.

4. "**Exception Unit Name**" option specifies unit name. Such as 'SysUtils', 'Unit1', 'Project1', etc. This option is optional. Filter will be applied only to exceptions raised from the specified unit.

Notes:
- This option uses ExceptionAddress property to determinate unit name.
- This option requires debug information to work.

5. "**Source Class Name**" option specifies class name. Such as 'TFileStream', 'TForm1', etc. This option is optional. Filter will be applied only to exception raised by methods of the specified class.

Notes:
- This option uses ExceptionAddress property to determinate class name.
- This option requires debug information to work.

6. "**Procedure/Method Name**" option specifies function or method name. Such as 'Create', 'Destroy', 'Read', 'Button1Click', etc. This option is optional. Filter will be applied only to exceptions raised by specified method or function.

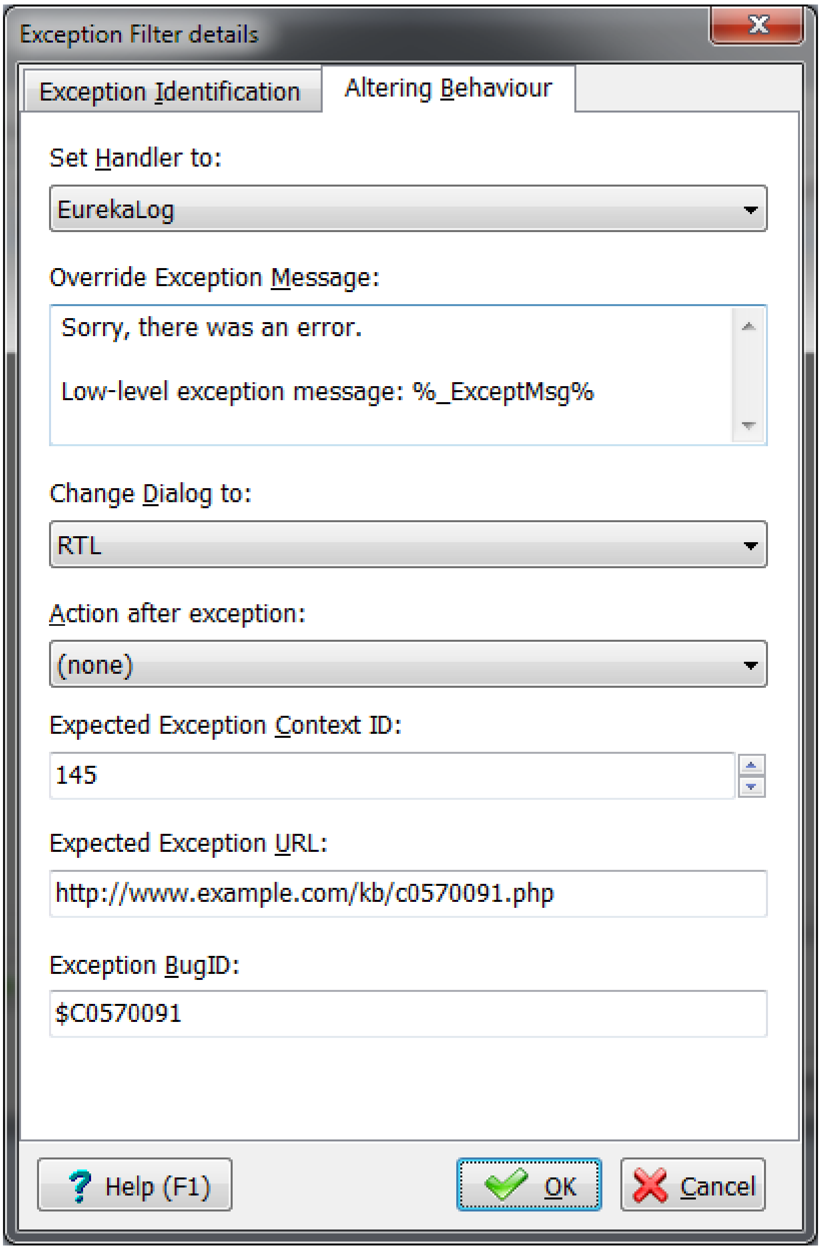
Notes:
- This option uses ExceptionAddress property to determinate function/procedure/method name.
- This option requires debug information to work.

7. "**Exception Properties**" option specifies set of properties to match. This option is optional. Filter will be applied to exceptions which have specified properties with exact the same values.

- Properties should be entered as: one property on single line.
- Each line must have "name=value" form (as indicated on the screenshot above).
- String values must be enclosed in double quotes.

- Use \q to insert double-quote inside string value.
- Enumerated types must be entered as integers.

Important note: this option can only work with Delphi/C++ Builder exceptions with RTTI info available! Standard exception classes usually do not have RTTI information.



Set Handler to: EurekaLog

Override Exception Message:

Sorry, there was an error.

Low-level exception message: %_ExceptMsg%

Change Dialog to: RTL

Action after exception: (none)

Expected Exception Context ID: 145

Expected Exception URL: http://www.example.com/kb/c0570091.php

Exception BugID: $C0570091

Altering Behavior Page

Altering Behavior Page contains filter options which alters exceptions. These options are applied to exception altering its properties and/or behavior. Options are applied only for matched filter.

Note: at least one of the options on this tab **must** be changed. Otherwise filter will do nothing.

Available options are:

1. "**Set Handler to**" option allow you to define who should handle exception:
- "None" handler will ignore exception. I.e. exception will not be handled at all. This handler

will not allow you to edit any additional options. This case is used for such exceptions as EAbort.
- "RTL" handler will pass exception to default processing. This is the same as if EurekaLog would be disabled. This handler will not allow you to edit any additional options. This case is used for "expected" exceptions, which do not require bug report creation.
- "EurekaLog" handler will pass exception to EurekaLog. This is default in EurekaLog-enabled application. This handler also allow you to specify additional options (see below). This is default and recommended case for most exception classes.

2. "**Override Exception Message**" option allow you to override exception message. Leave this field empty to use original exception message. You can use this option to supply user-friendly error message (such as "Sorry, there was an error. Please, restart application") instead of low-level error message (such as "Access violation in module Project1.exe at address 123456"). This option affects only visual dialogs. Bug report will contain original exception message. See also "Exception Message" option 244.

Note: you can use environment variables in this field, so you can also insert original exception message as part of your customized message (for example: "Sorry, there was an error. Please, restart application. Low-level error message: %_ExceptMsg%").

3. "**Change Dialog to**" option allow to change default dialog (as set in EurekaLog project options) to any available dialog class.

4. "**Action after exception**" option allow you to perform restart action after processing exception. This is useful for exceptions like access violation. You may want to terminate/restart your application after such exceptions to avoid induced exceptions.

5. "**Expected Exception Context ID**" option allow you to assign Context ID to exception. This is integer value which is expected to be ID of help topic describing exception.

Notes:
- Setting this option to any positive value (or -1) will convert exception to "expected" exception. Expected exception display error dialogs, but does not generate bug reports. 0 (default) will not mark exception as "expected".
- Set this property to a valid ID of help topic. If you do not have help topic for exception, but want to mark exception as "expected" - then set this option to -1.
- Use either "**Expected Exception Context ID**" option or "**Expected Exception URL**" option, but not both.
- Expected exception with valid Context ID (positive) or valid URL will cause "Help" button to be shown in error dialogs.

6. "**Expected Exception URL**" option allow you to assign URL for the exception. This URL is expected to point to online help or Knowledge Base topic/article describing exception.

Notes:
- Setting this option to any non-empty value will convert exception to "expected" exception. Expected exception display error dialogs, but does not generate bug reports.
- Set this property to a valid URL of online help topic. If you do not have help topic for exception, but want to mark exception as "expected" - then use "**Expected Exception Context ID**" option and set it to -1, keep "**Expected Exception URL**" option empty.
- Use either "**Expected Exception Context ID**" option or "**Expected Exception URL**" option, but not both.
- Expected exception with valid Context ID (positive) or valid URL will cause "Help" button to be shown in error dialogs.

7. "**Exception BugID**" option allow you to specify fixed BugID for this exception. If this value is 0 - BugID will be generated automatically. If this value is not 0 - automatic generation will be disabled. Specified BugID will be used.

Note: you can use both decimal (1234567890) and hexadecimal ($C7D40000) forms.

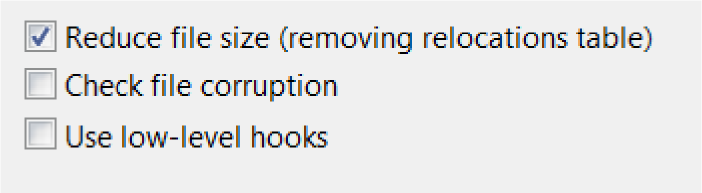
Warning: be extra careful when assigning this value. Your value may overlap with automatically generated BugIDs. Best strategy is to utilize user part of BugID (low word). See BugID 421 for more info.

See also:

- Configuring exception filters 343
- Using exception filters 185
- BugID 421

10.3.7.2 Build options page

This is "Advanced/Build options" page in EurekaLog project's options 225.



☑ Reduce file size (removing relocations table)

□ Check file corruption

□ Use low-level hooks

Build options

This page allows you to specify additional project build options.

1. "**Reduce file size**" option removes relocation table from file. This reduces file's size for about 10% (however, enabling EurekaLog also increases file's size).

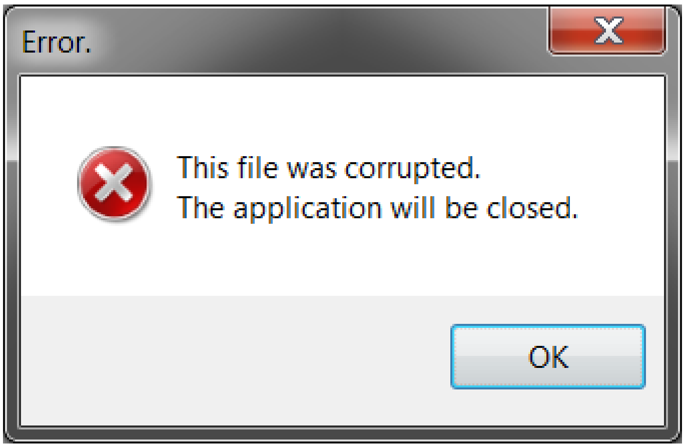
It's recommended to always keep this option on.

Note: this option have no effect for DLL and packages.

> **Technical explanation**
> When you compile a DLL (or a package which is a Delphi-specific DLL in disguise), the linker includes what is known as a relocation table. This table includes information about what addresses must be fixed up by the OS loader in the (likely) event that the DLL must be loaded at a different address than its intended-at-compile/link-time base address. You see, all DLLs come with a base address that is the "ideal" loading address of that module. The OS will try to load the DLL at this address to avoid the overhead of runtime rebasing (patching the in-memory pages of the DLL forces it to be paged in from disk and prevents cross-process sharing of the DLL pages). That's why you should set the Image base option in the Linker page of the project options of DLL and package projects. The default Image base that Delphi uses is $00400000 for both applications, DLLs and packages - and thus by default all DLLs and packages will need to be rebased - as that address is reserved for the process' EXE file.
>
> The implication is that an EXE file will always be loaded at the fixed virtual address $00400000 and that it will never need to be rebased. Alas, it doesn't really need its relocation table and we can safely remove it, shrinking the size of the .EXE without affecting its behavior or performance.

See also: External tools options page 259.

2. "**Check file corruption**" option adds check for file corruption in your project. If you enable this option, EurekaLog will calculate a CRC checksum of the compiled file and store it inside file. EurekaLog will also read this checksum from file on its startup (launch). If your executable was modified, EurekaLog will display an appropriate message and shutdown your application immediately:



Error.

This file was corrupted.
The application will be closed.

EurekaLog detected changes in executable file

You can use this option to ensure that your code wasn't modified.

Turn on for additional checks.
Turn off for best performance.

Warning: do not enable this option, if you're going to digitally sign your executable, to protect it with executable protector, or to pack it with executable packer. All 3 cases are not compatible with this option. Moreover, its use is redundant: each of 3 actions contain their own analogs of EurekaLog's "Check file corruption" option, so this option is not needed at all.

Notes:

- `CheckSum` field in the `IMAGE_OPTIONAL_HEADER` structure is used to store CRC value inside executable file;
- this option checks file on disk, not running process image;
- enabling this option will slow down loading and startup times on your executable. The bigger your executable file will be - the larger will be startup time: because the entire file must be read at startup.

See also: External tools options page 259.

3. "**Use low-level hooks**" option allows or forbids using of low-level hooks.

Using low-level hooks allows you to capture low-level information such as CPU state. Low-level hooks are also required for additional WER functionality. However, a documented way of installing low-level hook is available only in Windows XP and later. For older OS - undocumented hack will be used. If this option is unchecked - EurekaLog will use RTL functionality and will not install low-level hooks.

Note: low-level hooks will always be used on Delphi 2007 and below, since RTL support for handling exception was introduced in Delphi 2009.

Note: using of low-level hooks may introduce compatibility issues with 3rd party protection software.

Note: EurekaLog uses different implementation on Windows 2000 and Windows XP and above:

- Windows 2000: use SEH - inject hook into `KiUserExceptionDispatcher` (undocumented hack).
- Windows XP+: use VEH - add handler via documented API.

Note: this option controls only collecting information stage ("raise"). This option has no effect on other places. For example, hooks for handling exception are controlled by these options 352.

Turn on for best detalization.

Turn off for best compatibility.

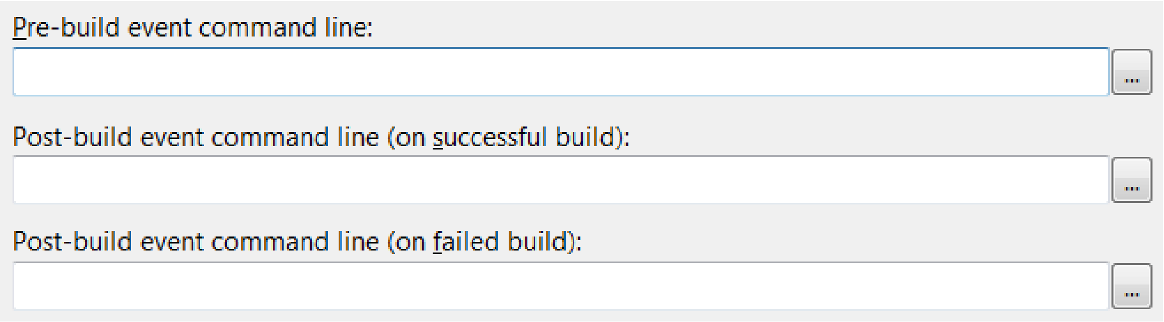
See also: External tools options page 259.

See also:

- Code hooks 352
- Build events 351
- Advanced options 341
- External tools options page 259
- Using EurekaLog with external software 514

10.3.7.2.1 Build events page

This is "Advanced/Build events" page in EurekaLog project's options 225.



Pre-build event command line:

Post-build event command line (on successful build):

Post-build event command line (on failed build):

Build events options

This page allows you to specify external application to be executed during application's build process. See also: External tools options page 259.

1. "**Pre-build event command line**" option specifies command that is to be performed before the project build starts.

2. "**Post-build event command line (on successful build)**" option specifies command that is to be performed after the build has successfully completed.

3. "**Post-build event command line (on failed build)**" option specifies command that is to be performed after the build has failed.

Notes:

- You can use environment variables 413 to customize command-line with variable parts. For example, to specify final exe file name. The following additional pseudo-variables are available to be used in build events:

_IDEProject - full file name of project file. Example: `C:\Project\Project.dproj`
_IDESource - full file name of source code file of project. Example: `C:\Project\Project.dpr`
_IDEConfig - full file name of EurekaLog options file. Example: `C:\Project\Project.dproj` or `C:\Project\Project.eof`
_IDETarget - full file name of final executable. Example: `C:\Project\Win32\Debug\Project.exe`
_IDESrc - file name only of the project. Example: `Project.dproj`
_IDEDst - output folder (with trailing path delimiter). Example: `.\Win32\Debug\`

- Don't forget that you need to enclose variable name in %, for example: **`%_IDEProject%`**.
- Don't forget to use " for file names with spaces within. Since you don't know exact values for variables - we strongly recommend to enclose variable names in quotes, for example: **`"%_IDEProject%"`**.
- Relative file paths are relative to project's folder.
- A common mistake is to confuse success/failure post-build events.
- These actions will be executed only when building application with EurekaLog. If you build your application with standard ways and then post-process result with EurekaLog - these actions will not be executed.

Delphi 2009+: IDE has similar options. If you're using EurekaLog in RAD Studio 2009 IDE or newer - it's recommended to use IDE's build events instead of EurekaLog's events. IDE events will be executed always, EurekaLog's events will be executed only when you compile your project with EurekaLog's help.

Delphi 2009+: order of actions during project's compilation is as follows:
1. EurekaLog pre-build event
2. IDE pre-build event
3. Project compile and link
4. IDE post-build event
5. EurekaLog link (post-processing)
6. EurekaLog post-build event

Note: you can insert a post-processing call to ecc32/emake 426 to IDE's post-build event. EurekaLog IDE expert will skip its own post-processing for already processed executables.

See also:
- IDE build events
- Build options 349
- Advanced options 341
- Environment variables 413
- External tools options page 259
- Using EurekaLog with external software 514

10.3.7.3 Code page

This is "Code" page in EurekaLog project's options 225.

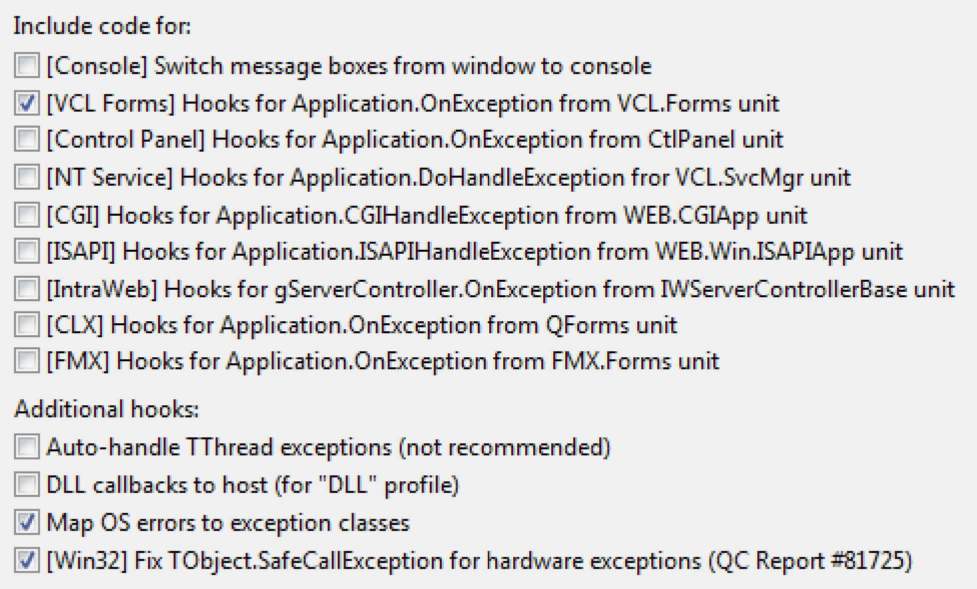
These are options for EurekaLog's code customizations. Each sub-category offers options to include or remove part of EurekaLog's code for executable.

The sub-categories are:
- Hooks 352
- Dialogs 354
- Debug information 355
- Send methods 355

See also:
- Configuring call stack 48

10.3.7.3.1 Hooks page

This is "Advanced/Code/Hooks" page in EurekaLog project's options 225.



Include code for:

☐ [Console] Switch message boxes from window to console
☑ [VCL Forms] Hooks for Application.OnException from VCL.Forms unit
☐ [Control Panel] Hooks for Application.OnException from CtlPanel unit
☐ [NT Service] Hooks for Application.DoHandleException for VCL.SvcMgr unit
☐ [CGI] Hooks for Application.CGIHandleException from WEB.CGIApp unit
☐ [ISAPI] Hooks for Application.ISAPIHandleException from WEB.Win.ISAPIApp unit
☐ [IntraWeb] Hooks for gServerController.OnException from IWServerControllerBase unit
☐ [CLX] Hooks for Application.OnException from QForms unit
☐ [FMX] Hooks for Application.OnException from FMX.Forms unit

Additional hooks:

☐ Auto-handle TThread exceptions (not recommended)
☐ DLL callbacks to host (for "DLL" profile)
☑ Map OS errors to exception classes
☑ [Win32] Fix TObject.SafeCallException for hardware exceptions (QC Report #81725)

Hooks code options

Each option includes hooks for specific cases (i.e. Pascal units). You should enable option to install hook (and invoke EurekaLog) or disable option to remove hook. See also: External tools options page 259.

- "**Console**" - for console applications.
- "**VCL Forms**" - for applications with Forms or VCL.Forms unit.
- "**Control Panel**" - for applications with CtlPanel unit.
- "**NT Service**" - for applications with SvcMgr unit.
- "**CGI**" - for applications with CGIApp unit.
- "**ISAPI**" - for applications with ISAPIApp unit.
- "**IntraWeb**" - for applications with IntraWeb units.
- "**CLX**" - for applications with QForms unit.
- "**FMX**" - for applications with FMX.Forms unit.

Usually these settings are set by selecting type of your application 363. You rarely need to change them.

Additional hooks are not tied to specific application type and may be used in any application. However, all of these options installs code hook - thus, they may be incompatible with some EXE cryptors, packers, protectors. Additional hooks may be customized by you depending on your needs.

1. "**Auto-handle TThread exceptions**" - option enables backward-compatible EurekaLog 6 behavior for threads. When you enable this option - EurekaLog will automatically handle exception in TThread. Default behavior is not to handle exception, but allow it to be saved in TThread.FatalException property, which can be analyzed/handled by caller thread.

Warning: enabling this option may result in multiple error dialogs at the same time (if several exception occur in multiple threads).

Note: it's not recommended to use this option. You should implement proper error handling for threads instead.

2. "**DLL callbacks to host**" option allows DLL to use exception manager from main module. This option is used with "DLL" profile when you need to call methods from exception manager (since there is no exception manager in DLL).

Warning: do not use this option for "Standalone DLL" profile or any other profiles except "DLL".

This options is turned on automatically for DLL profile. Usually you don't need to manage it manually.

This option can be used without EurekaLog in current module.

For more information: see using EurekaLog with DLLs 484.

3. "**Map OS errors to exception classes**" option converts all exceptions of EOSError class to (new) exception classes (from EMapWin32 unit). For example, there will be EOSAccessDenied exception raised instead of EOSError with ErrorCode = 5. This option will replace exception classes globally in all application. Old filtering code (i.e. "on E: EOSError do") should still work, because all new exception classes descend from EOSError. You can use this option to create exception filters for OS errors.

It's recommended to keep this option enabled.

This option also can be used without EurekaLog in current module.

4. "**Fix TObject.SafeCallException for hardware exceptions**" option fixes bug from Quality Central bug report #81725.

> TObject.SafeCallException works only for Delphi exceptions. If there is a hardware exception raised in safecall-method (like access violation or div by zero) - TObject.SafeCallException will be ignored, instead the fixed code of E_UNEXPECTED ($8000FFFF) will be used.
>
> With current implementation it is impossible to alter this behaviour except of using ugly workarounds.
>
> The problem seems to be in System._HandleAutoException routine. This routine does not call TObject.SafeCallException if exception in question is not Delphi exception.

Enable this option for COM applications or any other applications which use safecall-exceptions.

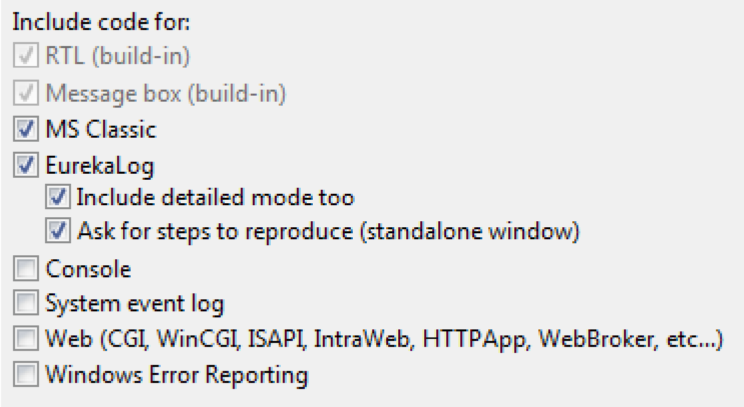
This option can be used without EurekaLog in current module.

Notes:
- This option has effect only for Win32 platform. It has no effect for 64bit code or MacOS.
- You probably would want to enable "Handle every SafeCall exception" option 341 for COM applications too.

See also:
- Application types 363
- External tools options page 259
- Using EurekaLog with external software 514

10.3.7.3.2 Dialogs page

This is "Advanced/Code/Dialogs" page in EurekaLog project's options 225.



Include code for:

☑ RTL (build-in)

☑ Message box (build-in)

☑ MS Classic

☑ EurekaLog

☑ Include detailed mode too

☑ Ask for steps to reproduce (standalone window)

□ Console

□ System event log

□ Web (CGI, WinCGI, ISAPI, IntraWeb, HTTPApp, WebBroker, etc...)

□ Windows Error Reporting

Dialogs code options

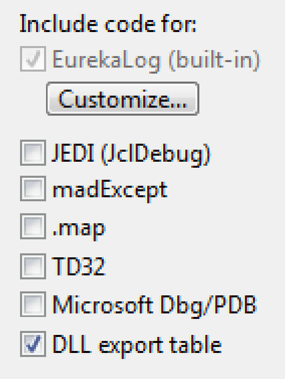
Each option includes code for specific dialog (i.e. Pascal unit). Usually these options are set automatically when you change dialog type 267. You can include additional dialogs if you want to change them from code (i.e. at run-time) or via exception filters.

See also:

- Available dialogs 370
- Configuring dialogs 267

10.3.7.3.3 Debug information page

This is "Advanced/Code/Debug information" page in EurekaLog project's options 225.



Include code for:

☑ EurekaLog (built-in)

□ JEDI (JclDebug)

□ madExcept

□ .map

□ TD32

□ Microsoft Dbg/PDB

☑ DLL export table

Debug information providers code options

Each option includes code for specific debug information providers (i.e. Pascal unit). You can change these options to include more providers in your application.

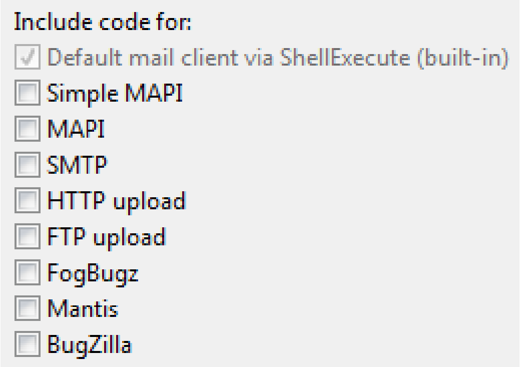
Note: support for madExcept is experimental.

See also:

- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
- Using EurekaLog with non-Embarcadero DLLs (Microsoft Visual Studio, etc.) 496
- Configuring call stack 48
- Using Microsoft DbgHelp DLL 504
- Debug information providers 409
- External tools options page 259
- Using EurekaLog with external software 514

10.3.7.3.4 Send engines page

This is "Advanced/Code/Send engines" page in EurekaLog project's options 225.



Include code for:

☑ Default mail client via ShellExecute (built-in)
□ Simple MAPI
□ MAPI
□ SMTP
□ HTTP upload
□ FTP upload
□ FogBugz
□ Mantis
□ BugZilla

Send engines code options

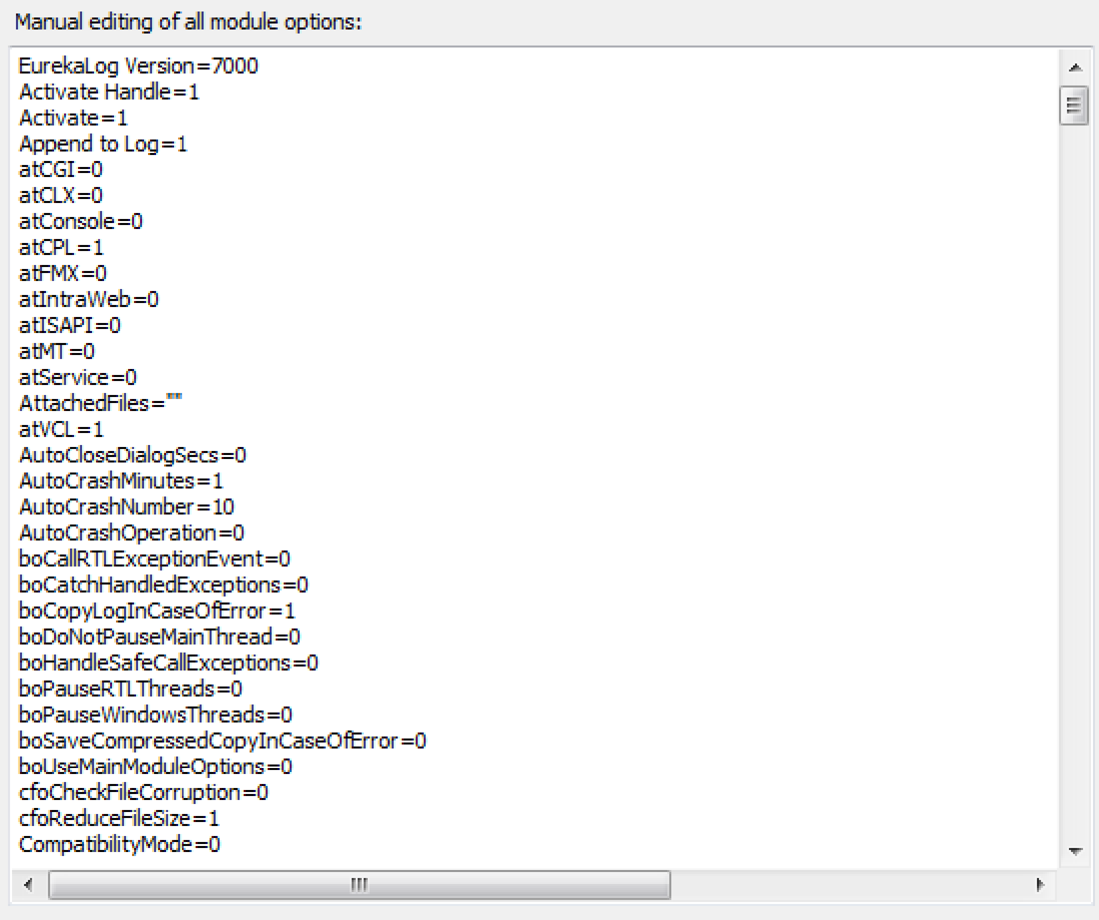
Each option includes code (i.e. Pascal unit) for specific send engine. Usually these options are set automatically when you change send options 304. You can include additional send engines if you want to change them from code at run-time.

See also:

- Send methods 390

10.3.7.4 Custom/Manual page

This is "Custom/Manual" page in EurekaLog project's options 225.



Manual editing of all module options:

```
EurekaLog Version=7000
Activate Handle=1
Activate=1
Append to Log=1
atCGI=0
atCLX=0
atConsole=0
atCPL=1
atFMX=0
atIntraWeb=0
atISAPI=0
atMT=0
atService=0
AttachedFiles=""
atVCL=1
AutoCloseDialogSecs=0
AutoCrashMinutes=1
AutoCrashNumber=10
AutoCrashOperation=0
boCallRTLExceptionEvent=0
boCatchHandledExceptions=0
boCopyLogInCaseOfError=1
boDoNotPauseMainThread=0
boHandleSafeCallExceptions=0
boPauseRTLThreads=0
boPauseWindowsThreads=0
boSaveCompressedCopyInCaseOfError=0
boUseMainModuleOptions=0
cfoCheckFileCorruption=0
cfoReduceFileSize=1
CompatibilityMode=0
```

Options

This page allows you to manually edit, add or remove any EurekaLog project option. You can use this page to add new options to your project.

Warning: be extra careful when editing options manually. Options will be used "as is".

See EurekaLog options 442 for more details about rules of encoding options. Please note that [Exception Log] header should not be present on this page.

Notes:

- Options are sorted for your convenience, but new options can be entered and placed in any order. There is no need to preserve sort order.
- Options with names started with "_" will not be saved into executable. Those are design-time only options, they are saved in project options, but not injected into final executable. Examples of such options are _BugAppVersion, _BugID, etc. 302
- We suggest to use "Custom_" prefix for your own keys. EurekaLog will never have any option name, starting with "Custom_". Thus, your names will not collide with EurekaLog settings.
- You can retrieve any option at run-time via `CurrentEurekaModuleOptions.CustomField['Option-Name']`.

See also:

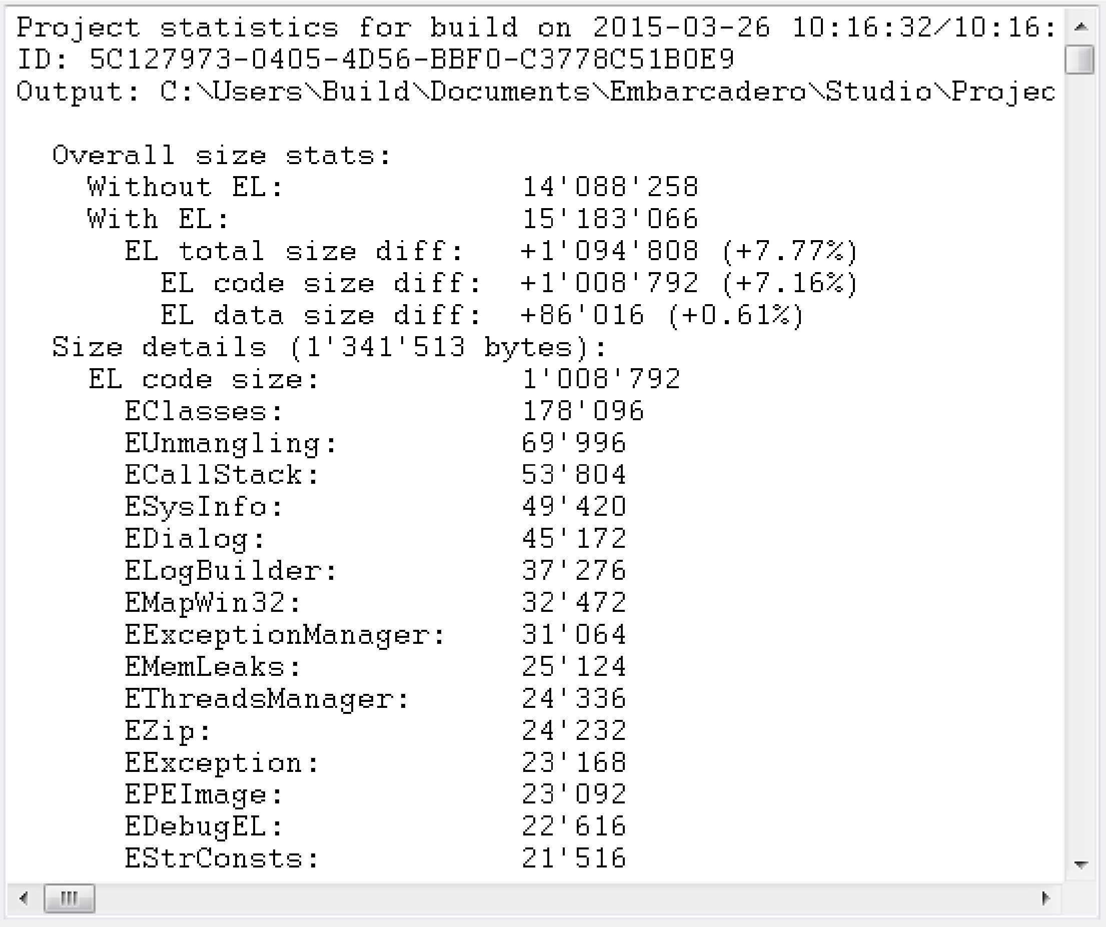
- Syntax for editing EurekaLog options 442
- Storing EurekaLog options 440
- Working with configurations 439

10.3.8 3rd party page

This page is reserved for 3rd party extensions of EurekaLog (EurekaLog's plugins).

10.3.9 Statistics

This is "Statistics" page in EurekaLog project's options 225.



```
Project statistics for build on 2015-03-26 10:16:32/10:16:
ID: 5C127973-0405-4D56-BBF0-C3778C51B0E9
Output: C:\Users\Build\Documents\Embarcadero\Studio\Projec

  Overall size stats:
    Without EL:              14'088'258
    With EL:                 15'183'066
      EL total size diff:    +1'094'808 (+7.77%)
        EL code size diff:   +1'008'792 (+7.16%)
        EL data size diff:   +86'016 (+0.61%)
  Size details (1'341'513 bytes):
    EL code size:            1'008'792
      EClasses:              178'096
      EUnmangling:           69'996
      ECallStack:            53'804
      ESysInfo:              49'420
      EDialog:               45'172
      ELogBuilder:           37'276
      EMapWin32:             32'472
      EExceptionManager:     31'064
      EMemLeaks:             25'124
      EThreadsManager:       24'336
      EZip:                  24'232
      EException:            23'168
      EPEImage:              23'092
      EDebugEL:              22'616
      EStrConsts:            21'516
```

Project build stats

This page displays stats about last project build.

This is optional page. It is displayed only when viewing/editing project options in IDE. It will be hided if no project is opened or when viewing/editing options in standalone Settings Editor tool.

Note: the stats are not collected by default. Additionally, the stats are not saved when project is closed. You have to enable stats collection and rebuild the project to view stats: enable stats collection by checking "Calculate stats" option 234 (and, optionally, "Debug output" option) at "General" page. Then rebuild the project and view "Statistics" page.

Sample output of statistics:

```
Project statistics for build on 2015-03-26 10:16:32/10:16:54
  for Project1.dproj project.
ID: 5C127973-0405-4D56-BBF0-C3778C51B0E9
Output: C:\Projects\Win32\Debug\Project1.exe

  Overall size stats:
    Without EL:             14'088'258
    With EL:                15'183'066
      EL total size diff:   +1'094'808 (+7.77%)
        EL code size diff:  +1'008'792 (+7.16%)
        EL data size diff:  +86'016 (+0.61%)
  Size details (1'341'513 bytes):
    EL code size:           1'008'792
      EClasses:             178'096
      EUnmangling:          69'996
      ECallStack:           53'804
      ESysInfo:             49'420
      EDialog:              45'172
      ELogBuilder:          37'276
      EMapWin32:            32'472
      EExceptionManager:    31'064
      EMemLeaks:            25'124
      EThreadsManager:      24'336
      EZip:                 24'232
      EException:           23'168
      EPEImage:             23'092
      EDebugEL:             22'616
      EStrConsts:           21'516
      ExceptionLog7:        19'804
      EResLeaks:            19'572
      EModules:             17'616
      EDialogWinAPIEurekaLogDetailed: 16'700
      ELogManager:          15'016
      ETools:               13'592
      EExceptionHook:       12'348
      EDialogWinAPIEurekaLog: 11'208
      EDebugInfo:           9'948
      ETypes:               9'516
      EHook:                9'444
      EXMLBuilder:          9'272
      EStackTracing:        9'068
      EDialogWinAPI:        8'644
      EInfoFormat:          8'388
      EDialogWinAPIMSClassic: 7'984
      EBase:                7'684
      EListView:            7'008
      EInject:              6'860
      ECompatibility:       6'716
      EConfig:              6'144
      ELowLevel:            5'968
      ESend:                5'832
      EEvents:              5'776
      EEncoding:            5'768
      EExceptionInfo:       5'724
      EExceptionInfoGeneric: 5'396
      EZLib:                4'948
      ELowLevelClasses:     4'884
      EAppType:             4'520
      ESpecificDelphi:      4'348
      EFreeze:              4'304
      ESendMailShell:       4'196
      EInternalDebug:       3'840
      EOSApiList:           3'820
      ECore:                3'748
      EWCTSupport:          3'672
      EDialogSendWinAPI:    3'504
      EHash:                3'436
      EDebugExports:        3'332
      EPNG:                 2'976
      EScreenshot:          2'768
      EDLLs:                2'492
      EDialogWinAPIStepsToReproduce: 2'064
      EFileMemory:          1'976
      EExceptionInfoDelphiUnicode: 1'760
      EDialogSend:          1'720
      EDisAsm:              1'636
      ENT:                  1'544
      EEncrypt:             1'516
      EAppVCL:              1'288
      ESendMail:            1'164
      EExceptionInfoDelphi2: 1'128
      EExceptionInfoDelphiANSI: 1'124
      EWCT:                 1'088
      EMLang:               584
      EFixSafeCallException: 428
      EMonitors:            344
      EPChars:              240
      EConsts:              20
    Debug info size:        332'721
      Uncompressed:         807'722
    Symbols size:           58
    Functions size:         4
    Stripped size:          -249'856
  Debug information details (807'722 bytes):
    Units:                  468
    Procedures:             22'009
    Lines:                  152'554
    Names:                  22'009
    1 byte (2-5-N):         116'676 (116'676 bytes)
    1 byte (3-3-P):         8'169 (8'169 bytes)
    2 bytes (7-5-V):        20'090 (40'180 bytes)
```

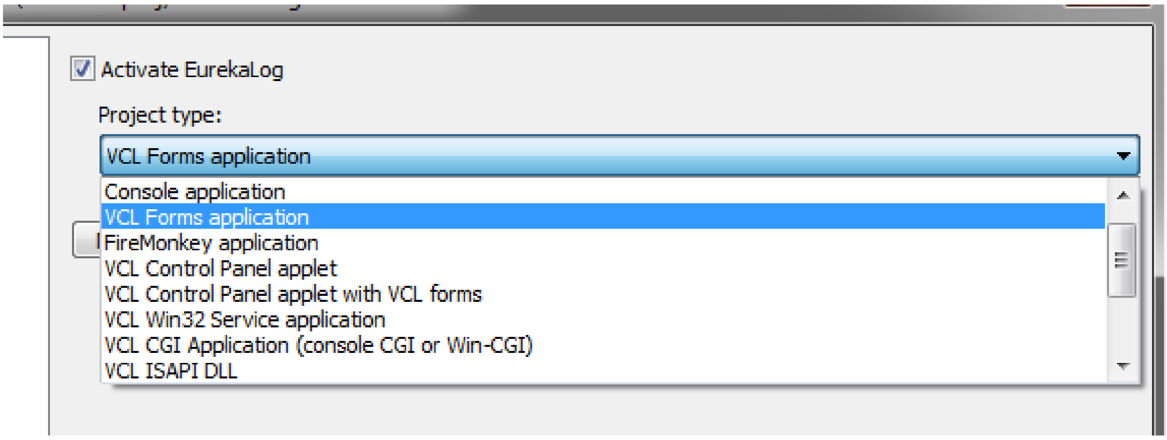
```
    4 bytes (16-12-V):      4'908 (19'632 bytes)
    8 bytes (16-16-V):      0 (0 bytes)
    16 bytes (32-32-V):     4'699 (75'184 bytes)
  Total time:               00:00:13.101
    Compilation time:       00:00:03.952
    Prepare time:           00:00:00.051
    Post-process time:      00:00:09.092
    Events time:            00:00:00.006
  Memory usage:
    Allocated:              103'242'467
    RAM:                    340'996'096
    Private:                338'440'192
    Virtual:                470'593'536

Analyzing file "C:\Projects\Win32\Debug\Project1.exe":

Target:                  x86-32

Module's version:        1.0.0.0
File size:               15183066
Module's description:
File creation:           2015-03-26 10:14:54
File last write:         2015-03-26 10:16:34
File last access:        2015-03-26 10:16:33
Compilation date:        2015-03-26 10:16:44

Is Borland image:        True
Is EurekaLog image:      True
Is JclDebug image:       False
Is MadExcept image:      False
Is TD32 image:           False
Is DWARF image:          False
Is Stab image:           False

Has .eldbg file:         False
Has .jdbg file:          False
Has .mad file:           False
Has .map file:           False
Has .tds file:           False
Has .dbg file:           False
Has .pdb file:           False

EurekaLog code version:  7.2.0.0 Enterprise
EurekaLog data version:  7.0.07
Code Machine ID:         D86FE1F598FB4242A796223D6909B720
Data Machine ID:         1EE7EC2155D37048A92392BEF05DABA5
Data Project ID:         7379125C0504564DBBF0C3778C51B0E9
EurekaLog's data size:   333143
  in % of original size: 2.24%

EurekaLog options:
Activate=1
atFixSafeCallException=1
atVCL=1
atWin32=1
CompatibilityMode=0
Debug=1
```

```
DeleteMapAfterCompile=1
Encrypt Password=""
EurekaLog Version=7007
idEurekaLog=1
idEurekaLogDetailed=1
idMSClassic=1
idStepsToReproduce=1
InjectCode=1
InjectInfo=1
InjectOptions=1
loEnableMMDebugMode=1
ProjectID="{5C127973-0405-4D56-BBF0-C3778C51B0E9}"
Stats=1
TextsCollection=""

EurekaLog symbols:
  ID: 11 (System.Classes.initialization), Address: 002CC550, Size: 144
  ID: 12 (System.Classes.finalization), Address: 00076A54, Size: 208
  ID: 13 (System.Variants.finalization), Address: 000579FC, Size: 192
  ID: 14 (System.SysUtils.initialization), Address: 002CC3CC, Size: 164
  ID: 15 (System.SysUtils.finalization), Address: 0004B9FC, Size: 404
  ID: 16 (System.finalization), Address: 0000B38C, Size: 104
  ID: 24 (InvokeRegistry.Init), Address: 0027A9F4, Size: 60
  ID: 25 (System.Win.ComObj.HandleSafeCallException), Address: 00136728, Size: 300

Module type:             exe

Sections:
  11536602 bytes (75%)   [004C6000] .debug (INIT DATA, READ)
  3105280 bytes (20%)    [00001000] .text (CODE, EXECUTE, READ)
  459264 bytes (3%)      [00455000] .rsrc (INIT DATA, READ)
  51200 bytes (0%)       [002FB000] .data (INIT DATA, READ, WRITE)
  17408 bytes (0%)       [0040F000] .idata (INIT DATA, READ, WRITE)
  8704 bytes (0%)        [002F8000] .itext (CODE, EXECUTE, READ)
  2560 bytes (0%)        [00414000] .didata (INIT DATA, READ, WRITE)
  512 bytes (0%)         [00417000] .rdata (INIT DATA, READ)
  512 bytes (0%)         [00415000] .edata (INIT DATA, READ)
  0 bytes (0%)           [00308000] .bss (READ, WRITE)
  0 bytes (0%)           [00416000] .tls (READ, WRITE)
  0 bytes (0%)           [00418000] .reloc (READ, WRITE)

Resources:
  333143 bytes (2%)      RCDATA ELDATA
  61223 bytes (0%)       GROUP_ICON MAINICON (x5)
  40356 bytes (0%)       STRINGTABLE  (x57)
  2724 bytes (0%)        RCDATA PACKAGEINFO
  2088 bytes (0%)        BITMAP EL_SEND
  1332 bytes (0%)        DIALOG EL_MS_DIALOG
  1320 bytes (0%)        BITMAP EL_DLL
  1320 bytes (0%)        BITMAP EL_NET
  714 bytes (0%)         MANIFEST #1
  618 bytes (0%)         DIALOG EL_DIALOG
  320 bytes (0%)         VERSION #1
  308 bytes (0%)         GROUP_CURSOR #32764 (x1)
  308 bytes (0%)         GROUP_CURSOR #32765 (x1)
  308 bytes (0%)         GROUP_CURSOR #32763 (x1)
  308 bytes (0%)         GROUP_CURSOR #32761 (x1)
  308 bytes (0%)         GROUP_CURSOR #32762 (x1)
  308 bytes (0%)         GROUP_CURSOR #32767 (x1)
  308 bytes (0%)         GROUP_CURSOR #32766 (x1)
  280 bytes (0%)         DIALOG EL_REQUEST
  248 bytes (0%)         BITMAP EL_MINUS
  248 bytes (0%)         BITMAP EL_PLUS
  240 bytes (0%)         RCDATA TFORM5
  232 bytes (0%)         BITMAP EL_PAS
  232 bytes (0%)         BITMAP EL_BPL
  232 bytes (0%)         BITMAP EL_VCL
  192 bytes (0%)         DIALOG EL_TAB_PROCESSESLIST
  192 bytes (0%)         DIALOG EL_TAB_MODULESLIST
  192 bytes (0%)         DIALOG EL_TAB_CALLSTACK
  188 bytes (0%)         DIALOG EL_TAB_CPU
  188 bytes (0%)         DIALOG EL_TAB_ASSEMBLER
  188 bytes (0%)         DIALOG EL_TAB_GENERAL
  182 bytes (0%)         DIALOG EL_SERVER
  16 bytes (0%)          RCDATA DVCLAL
  2 bytes (0%)           RCDATA PLATFORMTARGETS

Units:
  410980 bytes (2%)      System.Classes (System.Classes.pas)
  388976 bytes (2%)      Vcl.Themes (Vcl.Themes.pas)
  300912 bytes (1%)      System.Rtti (System.Rtti.pas)
  178024 bytes (1%)      EClasses (EClasses.pas)
  130600 bytes (0%)      Vcl.Controls (Vcl.Controls.pas)
  121960 bytes (0%)      Vcl.Forms (Vcl.Forms.pas)
  109092 bytes (0%)      Vcl.Themes (System.Generics.Collections.pas)
  91912 bytes (0%)       System.SysUtils (System.SysUtils.pas)
  72796 bytes (0%)       Vcl.Graphics (Vcl.Graphics.pas)
  71724 bytes (0%)       System.Rtti (System.Generics.Collections.pas)
  69996 bytes (0%)       EUnmangling (EUnmangling.pas)
  69056 bytes (0%)       System.Classes (System.Generics.Collections.pas)
  62072 bytes (0%)       System.Classes (System.Generics.Collections.pas)
  57120 bytes (0%)       System (System.pas)

... // cut to save space

EurekaLog's units:       EAppType, EAppVCL, EBase, ECallStack,
EClasses,
ECompatibility, EConfig, EConsts, ECore, EDebugEL, EDebugExports,
EDebugInfo,
EDialog, EDialogSend, EDialogSendWinAPI, EDialogWinAPI,
EDialogWinAPIEurekaLog,
EDialogWinAPIEurekaLogDetailed, EDialogWinAPIMSClassic,
EDialogWinAPIStepsToReproduce, EDisAsm, EDisAsmX8632, EDisAsmX8632Defs,
EDLLs,
EEncoding, EEncrypt, EEvents, EException, EExceptionHook,
EExceptionInfo,
EExceptionInfoDelphi2, EExceptionInfoDelphiANSI,
EExceptionInfoDelphiUnicode,
EExceptionInfoGeneric, EExceptionManager, EFileMemory,
EFixSafeCallException,
EFreeze, EHash, EHook, EInfoFormat, EInject, EInterfaces,
EInternalDebug,
EListView, ELogBuilder, ELogManager, ELowLevel, ELowLevelClasses,
EMapWin32,
EMemLeaks, EMLang, EModules, EMonitors, ENT, EOSApiList, EPChars,
EPEImage,
EPNG, EResLeaks, EScreenShot, ESend, ESendMail, ESendMailShell,
ESpecificDelphi,
EStackTracing, EStrConsts, ESysInfo, EThreadsManager, ETools, ETypes,
EUnmangling, EWCT, EWCTSupport, ExceptionLog7, EXMLBuilder, EZip, EZLib

Classes:
  105184 bytes (0%)      EClasses.TEurekaModuleOptions
  35404 bytes (0%)       Vcl.Controls.TWinControl
  32080 bytes (0%)       Vcl.Forms.TCustomForm
  28504 bytes (0%)       EDialog.TBaseDialog
  25316 bytes (0%)       Vcl.Controls.TControl
  21832 bytes (0%)       Vcl.Forms.TApplication
  21352 bytes (0%)       EException.TEurekaExceptionInfo
  18492 bytes (0%)       System.Classes.TStream
  18304 bytes (0%)       Vcl.Themes.TArray
  17880 bytes (0%)       ECallStack.TEurekaBaseStackList
  17728 bytes (0%)       Vcl.Themes.TUxThemeStyle
  17704 bytes (0%)       System.Classes.TArray
  17232 bytes (0%)       Vcl.Themes.{System.Generics.Collections}TList
    <Vcl.Themes.TPair<System.string,Vcl.Themes.TStyleManager.TSourceInfo>>
  17088 bytes (0%)       System.Classes.TReader
  16992 bytes (0%)       System.Classes.{System.Generics.Collections}TList
    <System.Classes.TPair<System.string,System.Classes.TPersistentClass>>
  16976 bytes (0%)       Vcl.Themes.{System.Generics.Collections}TList
    <Vcl.Themes.TPair<System.string,Vcl.Themes.TSysStyleHookClass>>
  16936 bytes (0%)       System.Rtti.{System.Generics.Collections}TList
    <System.Rtti.TPair<System.TypInfo.PTypeInfo,System.string>>
  16928 bytes (0%)       System.Classes.{System.Generics.Collections}TList
    <System.Classes.TPair<System.Integer,System.Classes.IInterfaceList>>
  16656 bytes (0%)       Vcl.Themes.{System.Generics.Collections}TList
    <Vcl.Themes.TChildControlInfo>

... // cut to save space
```

10.4 Application types

What is an "application type"?

"***Application type***" is just a **template** of settings and code. You can specify application type for your project in EurekaLog's project options 234:



☑ Activate EurekaLog

Project type:

VCL Forms application

- Console application
- VCL Forms application
- FireMonkey application
- VCL Control Panel applet
- VCL Control Panel applet with VCL forms
- VCL Win32 Service application
- VCL CGI Application (console CGI or Win-CGI)
- VCL ISAPI DLL

Application type selector in EurekaLog project options

When you select a project type - some settings of your project will be changed accordingly to selected application's type. See list of types below for more information.

- "***Supported application type***" means that EurekaLog has predefined template "out-of-the-box".
- "***Unsupported application type***" means that EurekaLog doesn't have a template for this kind of application. You need setup it manually.
- ***"Custom application type"*** means that template was defined by developer (you).

Supported types

EurekaLog supports the following types of applications "out-of-the-box":

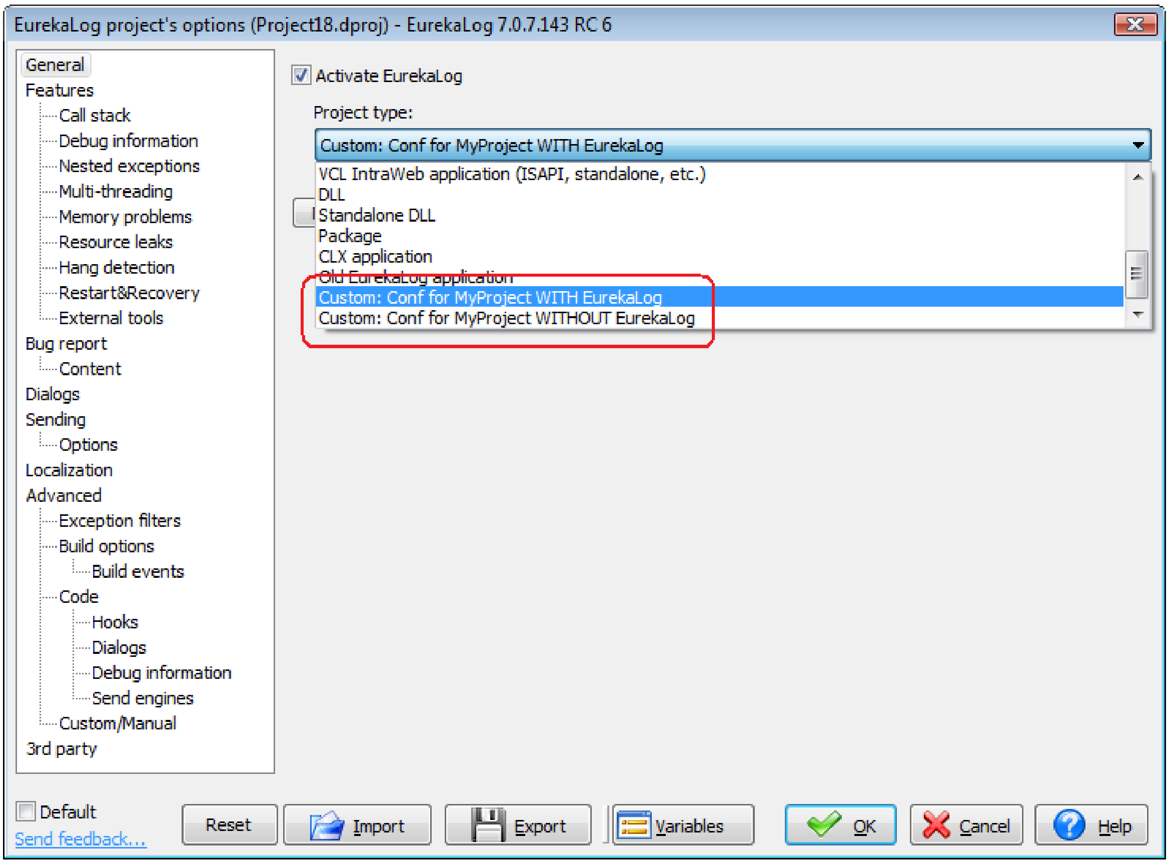
- VCL:
  - VCL GUI application 365 - this is for projects with Forms unit.
  - FireMonkey application 366 - this is for projects with FMX.Forms unit.
  - Control Panel application 366 - this is for projects with CplApplet unit. There is a second option for projects with both Forms and CplApplet units.
  - NT Service application 367 - this is for projects with SvcMgr unit.
  - CGI application 367 - this is for projects with CGIApp unit.
  - ISAPI application 367 - this is for projects with ISAPIApp unit.
  - IntraWeb application 368 - this is for projects with IntraWeb unit.
- Console application 368 - this is for console projects.
- DLL 368 - this is for DLL projects, which are used in EurekaLog-enabled application.
- Standalone DLL 369 - this is for the DLL projects, which are used in non-EurekaLog enabled application.
- Package 370 - this is for package projects.
- CLX application 366 - this is for projects with QForms unit.

Unsupported types

For other types of applications you need to select options manually. Just select "Custom settings / unsupported type" option and set the other necessary options. You also may need to write some code. See unsupported applications 370 for more information.

Custom types

You can set EurekaLog options and export them to external .eof file 227 (see also: Working with configurations 439). Custom configurations (i.e. saved in .eof files) will appear at the end of "Project type" combo-box, provided it was saved into default location (i.e. you did not change folder in "Export configuration" dialog):



EurekaLog project's options (Project18.dproj) - EurekaLog 7.0.7.143 RC 6

☑ Activate EurekaLog

Project type: Custom: Conf for MyProject WITH EurekaLog

"Project type" option shows two custom .eof files

Note: The path for loading custom configurations is %AppData%\Neos Eureka S.r.l \EurekaLog\Profiles\ (e.g. like C:\Users\UserName\AppData\Roaming\Neos Eureka S.r.l \EurekaLog\Profiles\).

Selecting any of "custom" values from "Project type" option will load configuration from corresponding .eof file into your project. All options in your project will be replaced with values from .eof file. Using custom configurations allows you to quickly switch between configurations in 2 mouse clicks.

See also:

- Unsupported application types 370
- Compiling your project with and without EurekaLog 445
- Working with configurations 439

10.4.1 VCL Forms Application

VCL Forms Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for Forms.TApplication.HandleException;
6. Include DLL exports debug information provider;
7. Include JCL debug information provider;
8. Dialog = MS Classic;
9. Include EurekaLog + EurekaLog Detailed dialogs;
10. Send report visual feedback: on;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL Forms applications. EurekaLog code and data will be injected into target EXE file.

10.4.2 CLX Forms Application

CLX Forms Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for QForms.TApplication.HandleException;
6. Include DLL exports debug information provider;
7. Include JCL debug information provider;
8. Dialog = MS Classic;
9. Include EurekaLog + EurekaLog Detailed dialogs;
10. Send report visual feedback: on;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled CLX Forms applications. EurekaLog code and data will be injected into target EXE file.

Note: this profile can only be used in Delphi 6-7 and C++ Builder 6.

10.4.3 FireMonkey application

FireMonkey Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for FMX.Forms.TApplication.HandleException;
6. Include DLL exports debug information provider;
7. Include JCL debug information provider;
8. Dialog = MS Classic;
9. Include EurekaLog + EurekaLog Detailed dialogs;
10. Send report visual feedback: on;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled FMX applications. EurekaLog code and data will be injected into target EXE file.

Note: this profile can only be used in Delphi/C++ Builder XE2 or above.

10.4.4 VCL Control Panel Application

VCL Control Panel Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for CtlPanel.TAppletApplication.HandleException;
6. Include DLL exports debug information provider;
7. Include JCL debug information provider;
8. Dialog = MS Classic;
9. Include EurekaLog + EurekaLog Detailed dialogs;
10. Send report visual feedback: on;

VCL Control Panel Application with VCL forms additionally includes:

11. Include hooks for Forms.TApplication.HandleException;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL Control Panel applications. EurekaLog code and data will be injected into target DLL file (.cpl).

Note: your project must be DLL, not exe.

10.4.5 VCL NT Service Application

VCL NT Service Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for SvcMgr.TApplication.HandleException;
6. Include DLL exports debug information provider;
7. Include JCL debug information provider;
8. Dialog = None;
9. Send report visual feedback: off;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL Win32/Win64 Services. EurekaLog code and data will be injected into target EXE file.

Note: it's recommended to use system logging dialog as explained in this article 535.

10.4.6 VCL CGI Application

VCL CGI Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for CGIApp.TApplication.HandleException;
6. Include hooks for HTTPApp.TCustomWebDispatcher.HandleException;
7. Include hooks for WebReq/WebBroker.TWebRequestHandler.HandleException;
8. Include DLL exports debug information provider;
9. Include JCL debug information provider;
10. Dialog = Web;
11. Send report visual feedback: off;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL web CGI-based applications. EurekaLog code and data will be injected into target EXE file.

10.4.7 VCL ISAPI Application

VCL ISAPI Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for ISAPIApp.TApplication.HandleException;
6. Include hooks for HTTPApp.TCustomWebDispatcher.HandleException;
7. Include hooks for WebReq/WebBroker.TWebRequestHandler.HandleException;

8. Include DLL exports debug information provider;
9. Include JCL debug information provider;
10. Dialog = Web;
11. Send report visual feedback: off;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL web ISAPI-based applications. EurekaLog code and data will be injected into target DLL file.

Note: your project must be DLL, not exe.

10.4.8 VCL IntraWeb Application

VCL IntraWeb Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include hooks for IWServerControllerBase.TIWServerControllerBase.DoException;
6. Include hooks for HTTPApp.TCustomWebDispatcher.HandleException;
7. Include DLL exports debug information provider;
8. Include JCL debug information provider;
9. Dialog = Web;
10. Send report visual feedback: off;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled VCL web IntraWeb applications. EurekaLog code and data will be injected into target EXE/DLL file.

10.4.9 Console Application

Console Application profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include DLL exports debug information provider;
6. Include JCL debug information provider;
7. Dialog = Console;
8. Send report visual feedback: on;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for standalone EurekaLog-enabled console applications. EurekaLog code and data will be injected into target EXE file.

Note: your application must be console application.

10.4.10 DLL

DLL profile includes the following options:

1. Add EurekaLog's code = False;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Dialog = None;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for DLL which is supposed to be used in any EurekaLog-enabled application. No EurekaLog code will be injected in this project. Only EurekaLog's data will be injected into target DLL file.

You should install host callbacks into DLL, if you want to handle exceptions in DLLs with creating EurekaLog bug reports - so DLL can call into host and ask it to create bug report. EurekaLog offers EAppDLL unit for this task (see Delphi example in this article 474). Non-Embarcadero DLLs should implement a similar service OR handle exceptions by themselves (without EurekaLog's assist).

Note: your project must be DLL, not exe.

Important notes:
1. DLL profile is designed to exclude EurekaLog code from executable. This means that most EurekaLog options will have no effect for a project with DLL profile. For example, dialog settings will be ignored, since there is no dialog code in DLL compiled with this profile. Dialog code will be in EurekaLog-enabled host. Thus, you should adjust options of the host. However, some options will still affect DLL. For example, application type hooks 352 (such as VCL forms, etc.), memory manager 250, compilation options 349, and debug information options 243.

2. Do not include EurekaLog units into project compiled with DLL profile. This is not supported and may have unexpected results. In particular, do not link DLL project with EurekaLogCore run-time package.

See also:
- Standalone DLL profile 369
- Using EurekaLog in DLLs 455
- Single instance of exception tracer 474

10.4.11 DLL (standalone)

Standalone DLL profile includes the following options:

1. Add EurekaLog's code = True;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Include DLL exports debug information provider;
6. Include JCL debug information provider;
7. Dialog = None;
8. Send report visual feedback: off;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for DLL which is supposed to be used in any non-EurekaLog-enabled application. EurekaLog code and data will be injected into target DLL file.

It's generally a good idea not to let exceptions escape DLLs 457 compiled with this profile. Otherwise a caller (.exe file) may not know how to properly release resources associated with exception from DLL.

Note: your project must be DLL, not exe.

See also:
- DLL profile 368
- Using EurekaLog in DLLs 455
- Multiple instances of exception tracer 480

10.4.12 Package

Package profile includes the following options:

1. Add EurekaLog's code = False;
2. Add module's options = True;
3. Add debug information = True;
4. Delete .map file after compilation = True;
5. Dialog = None;

(bold lines are mandatory for profile; normal lines are optional - they are set by default, but can be altered later)

This profile is for packages. No EurekaLog code will be injected in this project. Only EurekaLog's data will be injected into target BPL file. EurekaLog's code should reside in .exe file (recommended) or should be included as run-time package (EurekaLogCore.bpl).

Note: your project must be package (.bpl), not exe, not DLL.

10.4.13 Unsupported application types

"Unsupported" application type 363 doesn't really mean that you can not use EurekaLog with your application. It just means that EurekaLog doesn't have appropriate template for your application and you'll need to write some code. Nothing scary.

Basically, you need to find some kind of "OnException" event and invoke EurekaLog to handle current exception. For example, VCL and FMX applications have global Application object with OnException event. A console application can use explicit try/except block around code. A IntraWeb application has DoException event in ControllerBase. Many Indy objects have their own OnException events. And so on. Just refer to framework documentation to find the proper event for handling exceptions.

To perform exception handling (processing) - call ExceptionManager.ShowLastExceptionData function (EExceptionManager unit).

10.5 Dialogs

You can select an appropriate dialog type for your application 52.

There are many dialogs available:

Dialog	Where applicable	Description
(none) 371	Any application's type	Dialog that does nothing at all. It doesn't show anything.
RTL 371	Any application's type	Standard application's error dialog.
MessageBox 373	Visual applications only	Displays error message via Windows.MessageBox function.
MS Classic 377	Visual applications only	Displays error message in MS Windows XP-style dialog.
EurekaLog 379	Visual applications only	Displays error message in EurekaLog-style dialog.
Console 382	Console applications only	Displays error by outputting it to console (error output).
System log reporting 384	Any application's type	Outputs error message to system log (event log).
WEB 386	Web applications only	Displays error in returned HTML page.

Windows Error Reporting 389	Any application's type	Elevates error to OS.

You can know more about each dialog by clicking on each dialog in the table above.

No dialog 371 can be selected when you don't need any error reporting. Usually, in this case you save bug report to file 46 or send bug report to developer 53. Without dialog these actions will be performed automatically without need for user to act.

RTL dialog 371 invokes default application's error dialog after executing EurekaLog's processing (like saving bug report 46 and sending it 53). It's a good choice to add EurekaLog to your application without affecting any visual aspect.

MessageBox 373, MS Classic 377 and EurekaLog 379 dialogs can be used almost in any application - though it may be not a very good idea to use them in services or web applications (even though you may use it and sometimes it will even work). These dialog types usually works good in GUI or console applications.

System log reporting 384 is usually used in service or web applications. It records error message to system event log.

Console 382 and WEB 386 dialogs are special - they are applicable only to corresponding application's type. That's because other application types don't have a connected output for these dialogs. Console dialog write error message to std_error output. WEB dialog sends a HTML page with error to the client (when possible).

WER 389 (Windows Error Reporting) is a special dialog, which invokes default OS processing and shows system error dialog.

See also:

- Configuring dialogs 52
- Configuring dialog options 267

10.5.1 (none)

This is a "dialog" which means "do not show dialog at all".

It's usually used when you don't want to interact with user (like: non-interactive services or web-applications). If you set this dialog for your application then it will not show error dialog to the user, instead there will be a standard exception processing: like saving bug report 46 and/or sending report to developer 53.

This "dialog" will store an empty "steps to reproduce" text, uses default user e-mail and perform report sending (as if the user has clicked on "Send" button) and sends screenshot, if it was created (specified in send options 304).

Default user information is extracted via GetUserEMail, GetUserFullName and GetUserName functions. You can set default user e-mail by calling SetUserEmail function.

It's not recommended to use this dialog type for sending reports in application which interacts with user. That's because this dialog type sends report silently, without any confirmation from user.

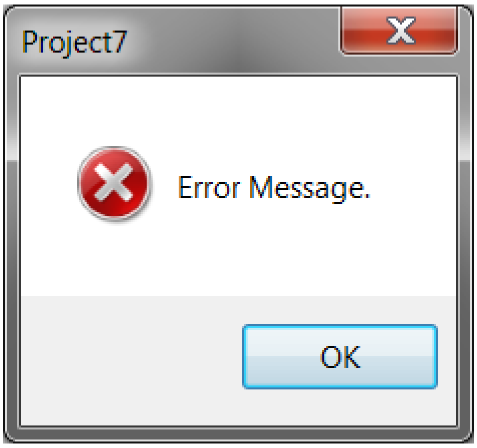
Constant: edtNone.

See also:

- RTL dialog 371
- Other dialogs 370

10.5.2 RTL

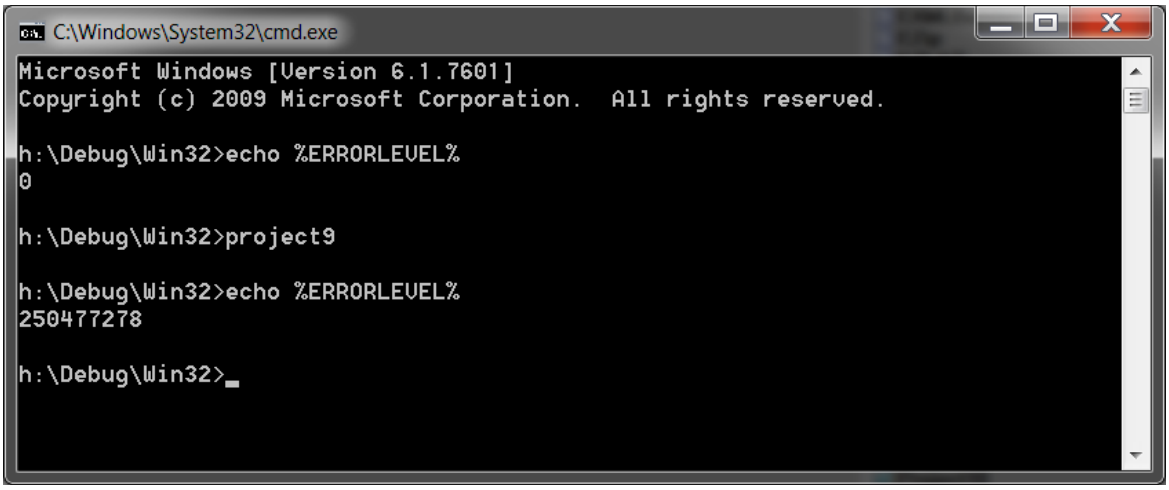
This is standard error dialog in your application. Exact visual appearance depends on your application's type. For example:



Project7

Error Message.

VCL Forms application: exception is handled by Application.ShowException



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>echo %ERRORLEVEL%
0

h:\Debug\Win32>project9

h:\Debug\Win32>echo %ERRORLEVEL%
250477278

h:\Debug\Win32>_
```

Console application: exception is not handled. Application terminates with $0EEDFADE error code.

It's usually used when you don't want to change existing visual experience at all, but want to add EurekaLog's capabilities to your application. If you set this dialog for your application then it will perform EurekaLog's tasks like saving bug report 46 and/or sending report to developer 53 and then invoke default error processing as if EurekaLog isn't here.

Note: Unlike any other dialogs, this dialog invokes EurekaLog's tasks first and only then invokes standard error dialog - that's because standard error processing may include application's termination, so reverse order will not invoke EurekaLog at all. Take this into account if you want to show messages about send status - they will be showed before error message itself.

This dialog will store an empty "steps to reproduce" text, uses default user e-mail and perform report sending (as if the user has clicked on "Send" button) and sends screenshot, if it was created (specified in send options 304).

Default user information is extracted via GetUserEMail, GetUserFullName and GetUserName functions. You can set default user e-mail by calling SetUserEmail function.

This dialog is not customizable. If you want a customized behaviour - you need to use one of EurekaLog's dialogs 370 instead.

Constant: edtRTL.

See also:

- (none) dialog 371

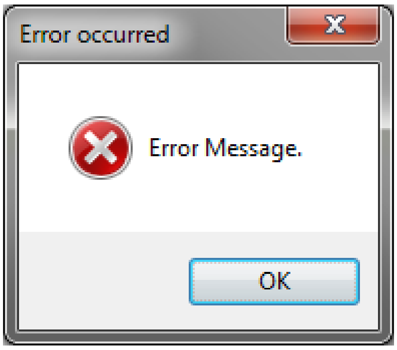
- WER dialog 389
- Other dialogs 370

10.5.3 Message box

This is a wrapper for Windows.MessageBox function. It is an efficient, simple and reliable dialog type.

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

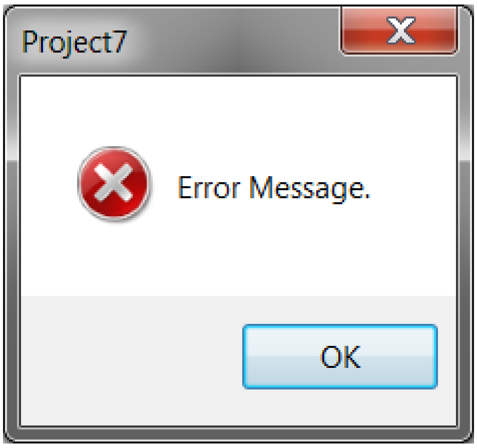
This dialog type displays error message in a popup window: the default message box. For example:



Error Message.

EurekaLog's MessageBox dialog in standard mode

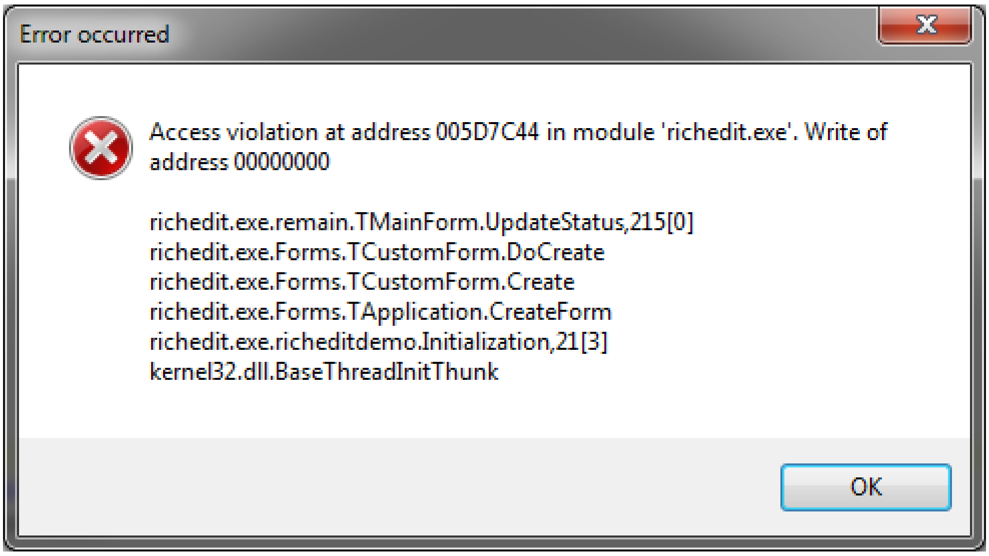
Compare it with the default error message in VCL Forms application (see also: RTL dialog 371):



Error Message.

Standard error message in VCL Forms application

You can enable "Detailed mode" to get a little more descriptive error message - by including a compact call stack:



Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

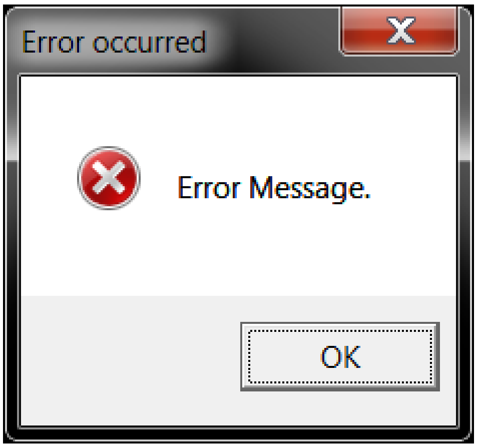
richedit.exe.remain.TMainForm.UpdateStatus,215[0]
richedit.exe.Forms.TCustomForm.DoCreate
richedit.exe.Forms.TCustomForm.Create
richedit.exe.Forms.TApplication.CreateForm
richedit.exe.richeditdemo.Initialization,21[3]
kernel32.dll.BaseThreadInitThunk

EurekaLog's MessageBox dialog in detailed mode

If you uncheck the "Use native message box" option - then the error message will always be displayed via Windows.MessageBox function. If you check the "Use native message box" option - then the error message will be displayed via EAppType.MessageBox function.

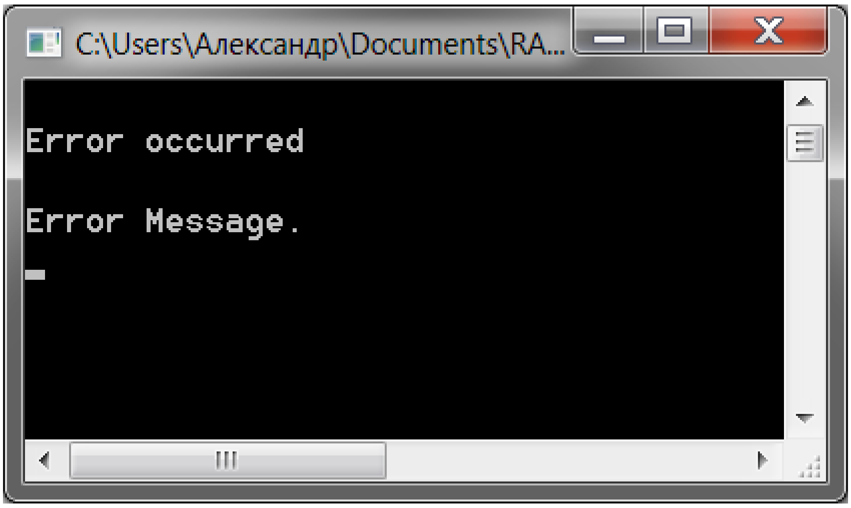
What does it mean? By default "native" style is the same Windows.MessageBox function. However, some types of application (currently it's a console and web) overrides this to custom routines. For example, "native" message box in console application - it's an output to console. A "native" message box for IntraWeb application - it's a scripted dialog (via WebApplication.ShowMessage).

Here is how it looks for console application:



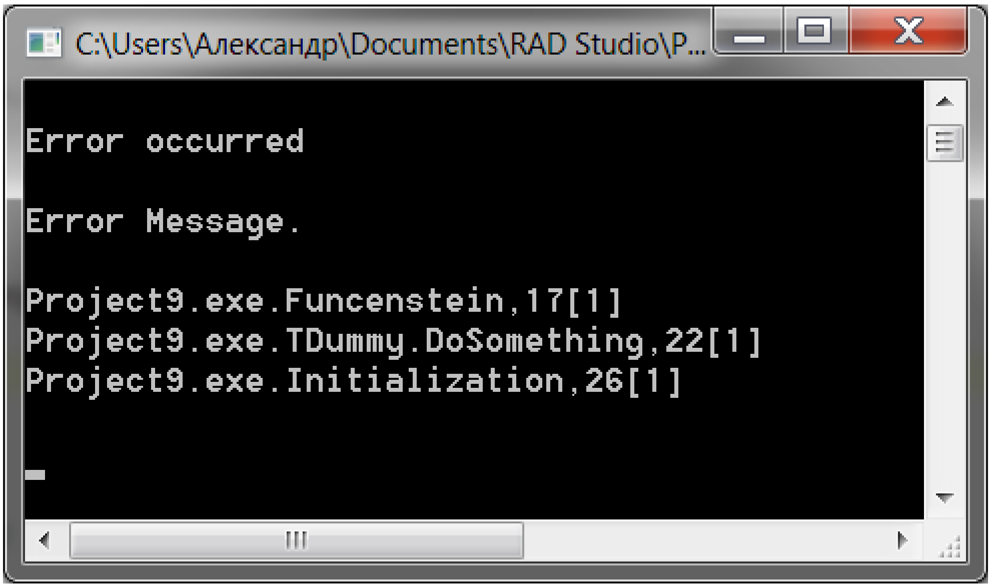
Error Message.

"Use native message box" is unchecked



```
Error occurred

Error Message.
```

"Use native message box" is checked



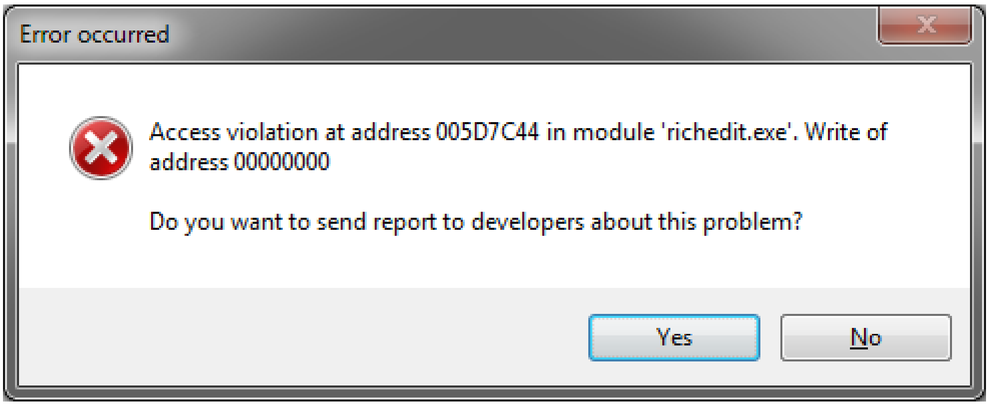
```
Error occurred

Error Message.

Project9.exe.Funcenstein,17[1]
Project9.exe.TDummy.DoSomething,22[1]
Project9.exe.Initialization,26[1]
```

Both "Use native message box" and "Detailed mode" are checked

Message box type dialog is never ask user for e-mail or "steps to reproduce". The default behaviour is to store empty "steps to reproduce", use default user e-mail and send screenshot, if it was created (specified in send options 304). Default user information is extracted via GetUserEMail, GetUserFullName and GetUserName functions. You can set default user e-mail by calling SetUserEmail function.

If you setup report sending 53 and enable "Ask user for send consent" option - then message box will ask user for their consent before sending bug report to developer - by showing question "Do you want to send report" and presenting "Yes" and "No" buttons.

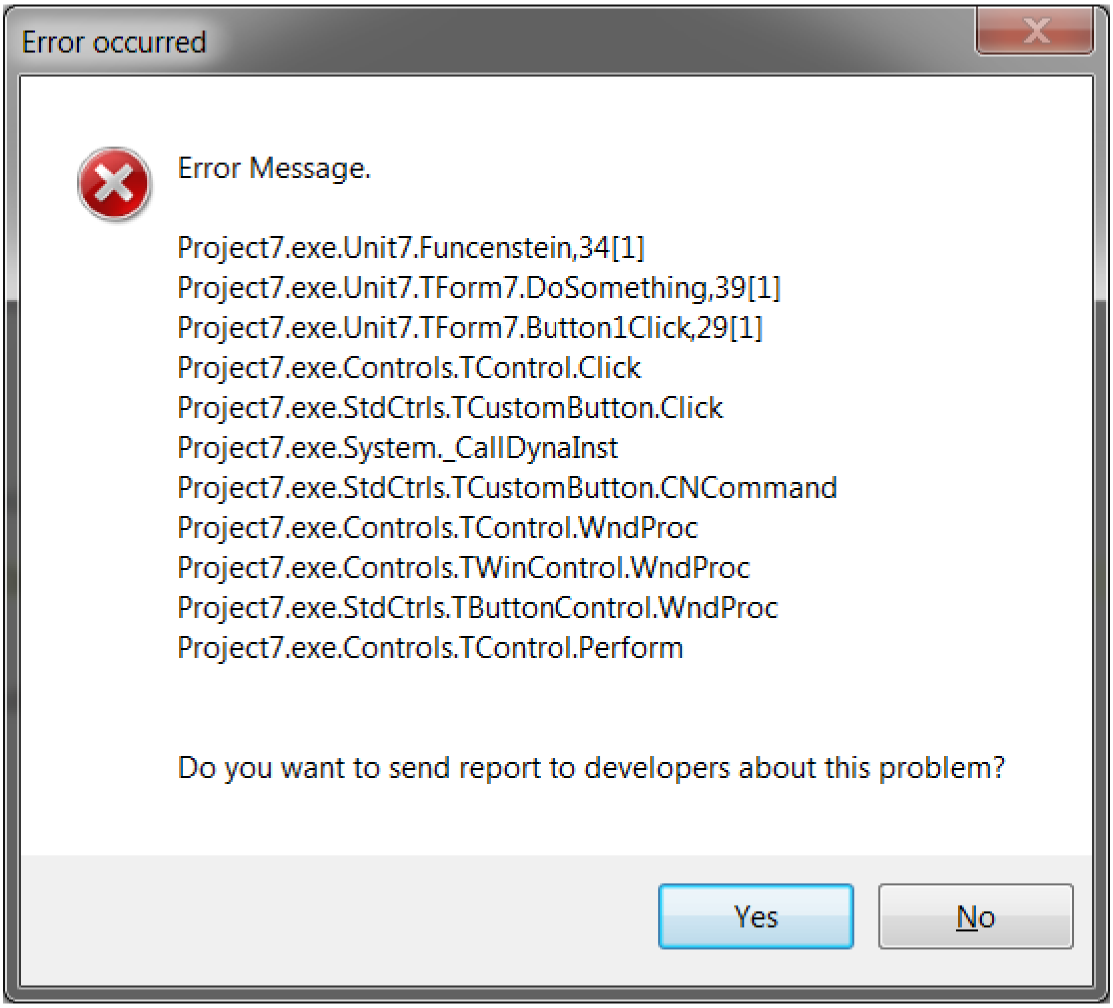
If "Ask user for send consent" option is unchecked - then message box will looks like above. If "Ask user for send consent" option is checked - message box will looks like this:



Access violation at address 005D7C44 in module 'richedit.exe'. Write of address 00000000

Do you want to send report to developers about this problem?

Non detailed mode with asking for consent

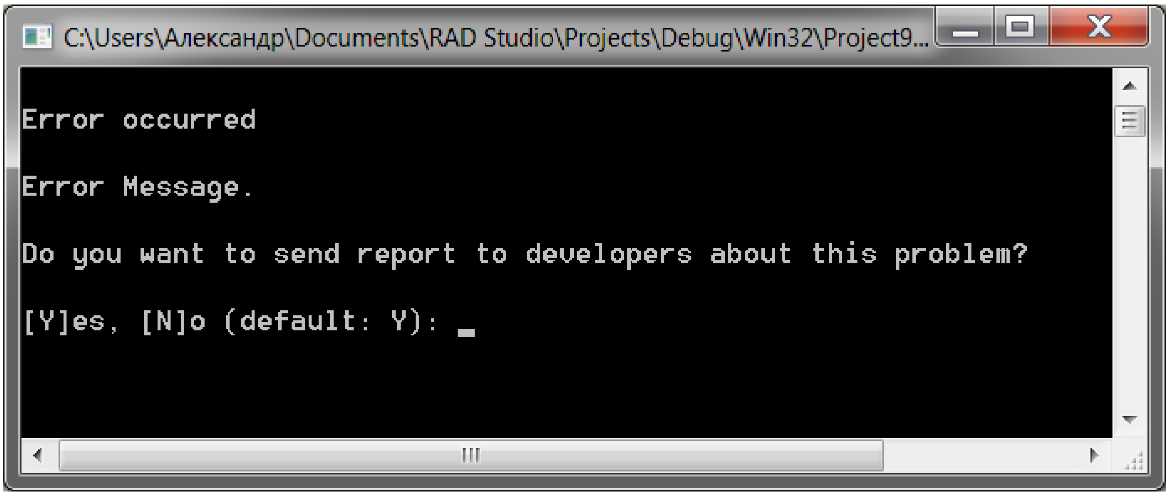


Error Message.

```
Project7.exe.Unit7.Funcenstein,34[1]
Project7.exe.Unit7.TForm7.DoSomething,39[1]
Project7.exe.Unit7.TForm7.Button1Click,29[1]
Project7.exe.Controls.TControl.Click
Project7.exe.StdCtrls.TCustomButton.Click
Project7.exe.System._CallDynaInst
Project7.exe.StdCtrls.TCustomButton.CNCommand
Project7.exe.Controls.TControl.WndProc
Project7.exe.Controls.TWinControl.WndProc
Project7.exe.StdCtrls.TButtonControl.WndProc
Project7.exe.Controls.TControl.Perform
```

Do you want to send report to developers about this problem?

Detailed mode with asking for consent


C:\Users\Александр\Documents\RAD Studio\Projects\Debug\Win32\Project9...

```
Error occurred

Error Message.

Do you want to send report to developers about this problem?

[Y]es, [N]o (default: Y): _
```

Non detailed console application with "native" message box and asking for consent

Default choice is selected by enabling/disabling "Default choice: send report" option. If this option is checked - the default choice is "send the report". If this option is unchecked - then the default choice is "do NOT send the report".

Note: what's the difference between console dialog type 382 and message box dialog type with "native" mode? The message box dialog tries to be as much similar to popup message box window as possible. I.e. it displays the same information and it asks for confirmation - pressing "Enter" in console is like clicking on "OK" button in the message box. On the other hand, console dialog type doesn't try to be similar to anyone. It displays its own set of information and it doesn't wait for user, until it needs user's answer.

Constants: edtMessageBox, edtMessageBoxDetailed.

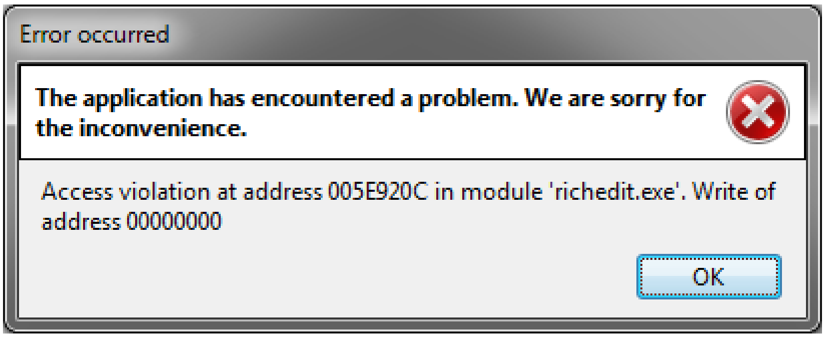
See also:
- MessageBox dialog options 268
- Console dialog 382
- MS Classic dialog 377
- EurekaLog dialog 379
- Other dialogs 370
- Nested (chained) exceptions 244

10.5.4 Windows Classic

This is a dialog type which is similar to classical Windows Error Reporting error dialog in Windows XP. This dialog is written with WinAPI (no VCL) and it's a preferred dialog in your GUI applications.

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

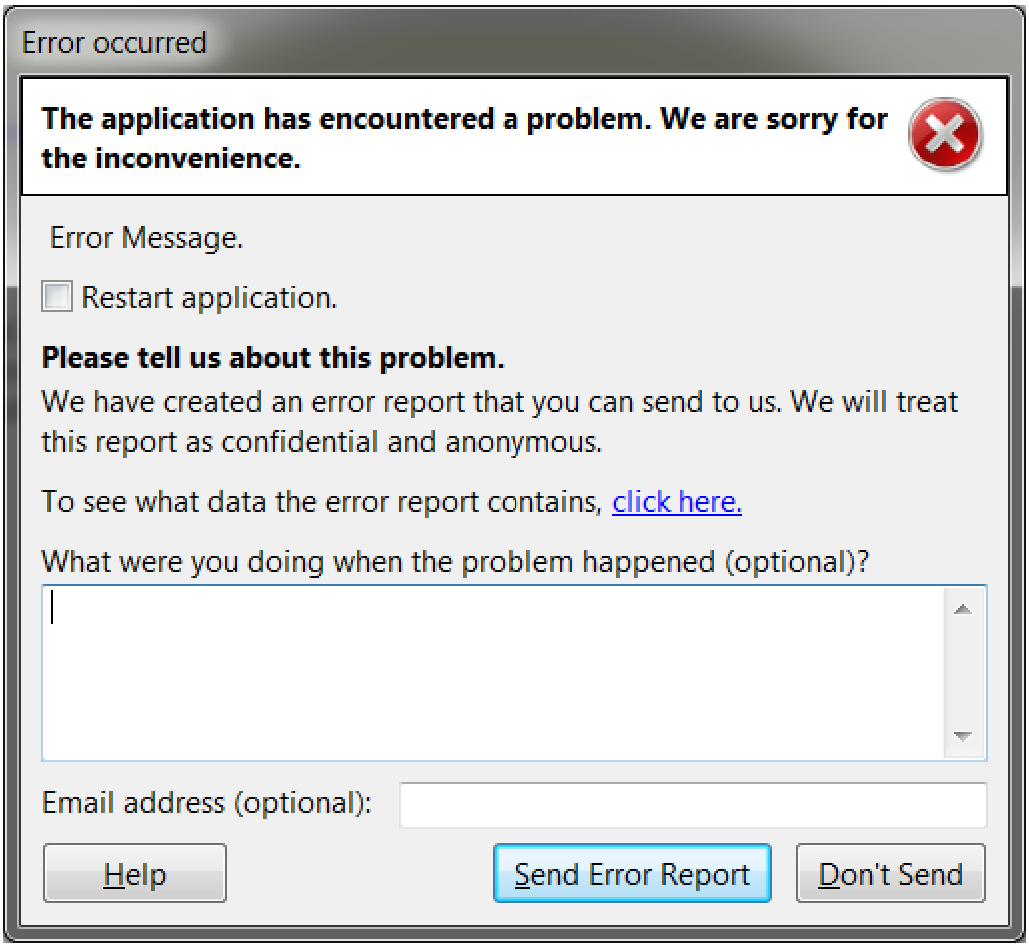
Dialog have wide range of options:



The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

Minimalistic view of MS Classic dialog type



Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Error Message.

□ Restart application.

Please tell us about this problem.

We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

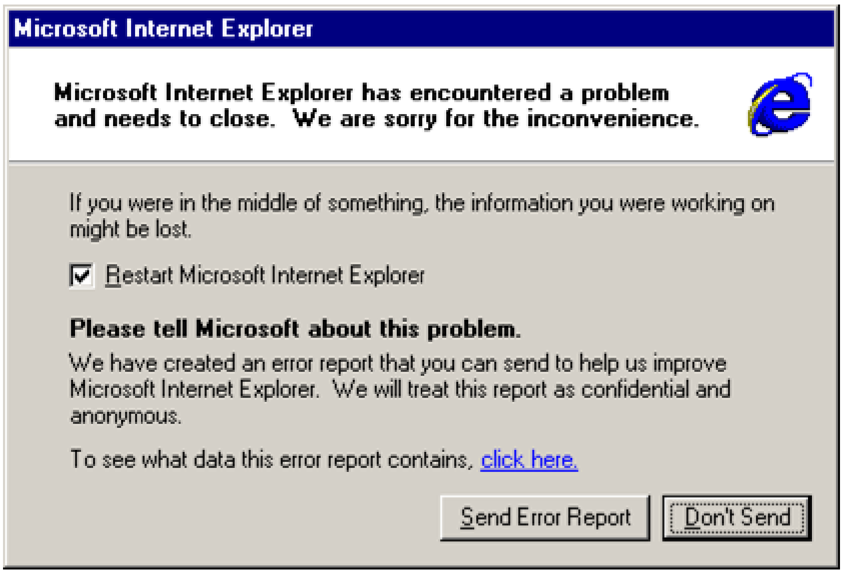
To see what data the error report contains, click here.

What were you doing when the problem happened (optional)?

Email address (optional):

MS Classic with all options checked

For comparison: here is both original Windows Error Dialog and EurekaLog dialog on Windows XP with classic themes:



Microsoft Internet Explorer

Microsoft Internet Explorer has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

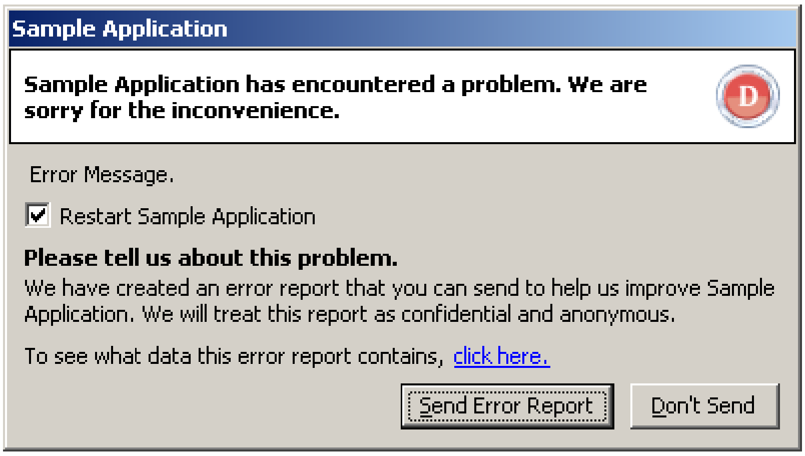
☑ Restart Microsoft Internet Explorer

Please tell Microsoft about this problem.

We have created an error report that you can send to help us improve Microsoft Internet Explorer. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Original MS error dialog



Sample Application

Sample Application has encountered a problem. We are sorry for the inconvenience.

Error Message.

☑ Restart Sample Application

Please tell us about this problem.
We have created an error report that you can send to help us improve Sample Application. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

EurekaLog's MS Classic style dialog in maximum compatibility mode

This dialog type sends screenshot, if it was created (specified in send options 304).

Constant: edtMSClassic.

See also:
- MS Classic dialog options 271
- EurekaLog dialog 379
- Other dialogs 370
- Nested (chained) exceptions 244

10.5.5 EurekaLog

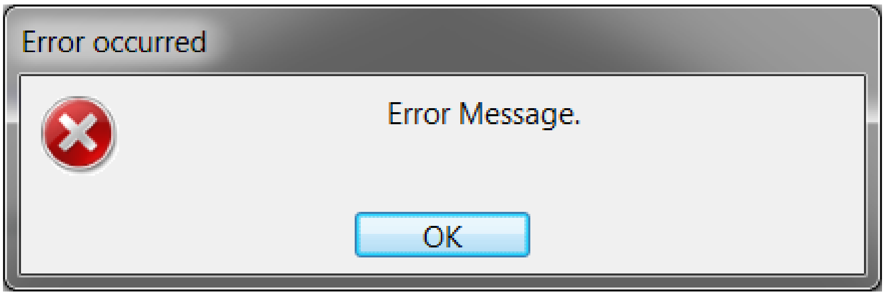
This is a dialog type which is visually similar to MessageBox 373, but have much more customizable options. This dialog is written with WinAPI (no VCL) and it's a preferred dialog in development environments, where you need much options.

The dialog also have so-called "detailed" form which displays bug report. See this article 72 for more information about bug reports.

It may be not a good idea to use it for end-user application, since this dialog is a bit too technical, so it may be not very user-friendly for non-advanced users.

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

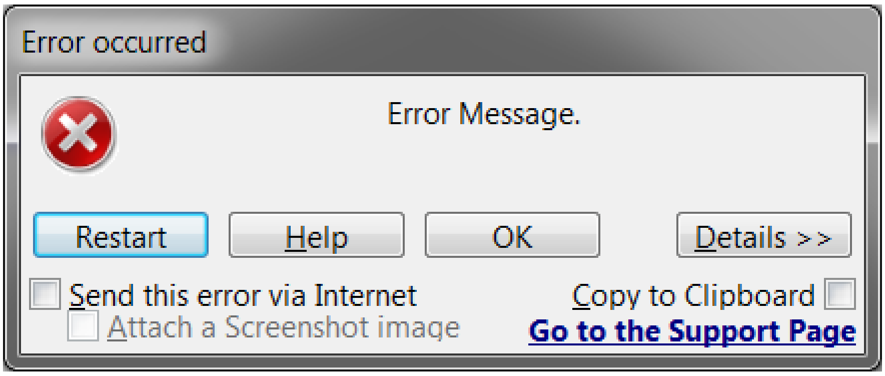
Dialog have wide range of options:



Error occurred

Error Message.

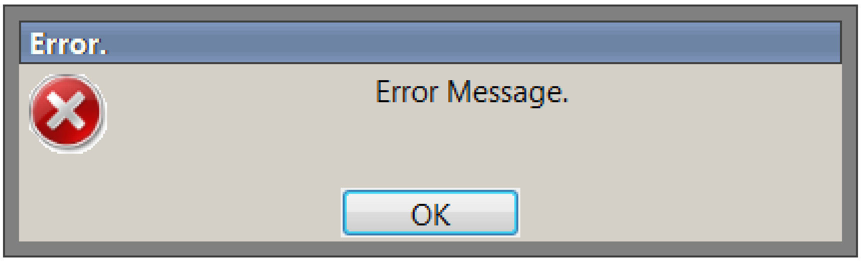
Minimalistic view of EurekaLog dialog



Error Message.

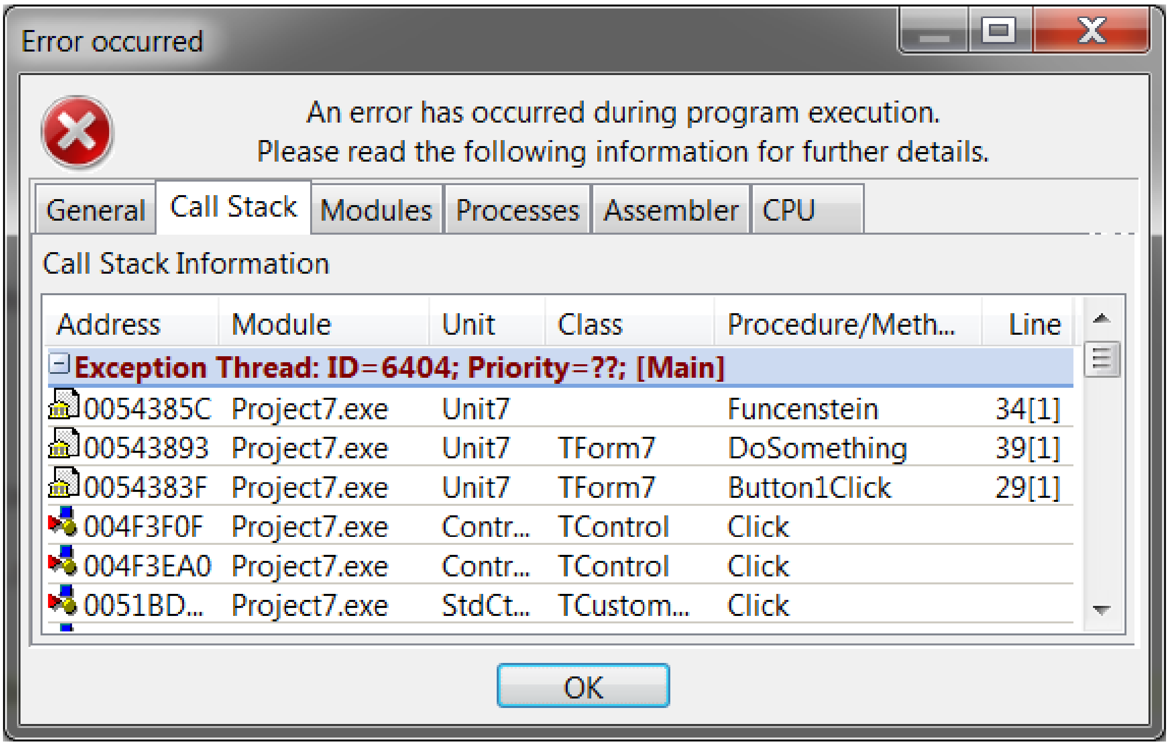
□ Send this error via Internet
□ Attach a Screenshot image
Copy to Clipboard □

EurekaLog dialog with all options checked



Error Message.

EurekaLog dialog with "EurekaLog style"

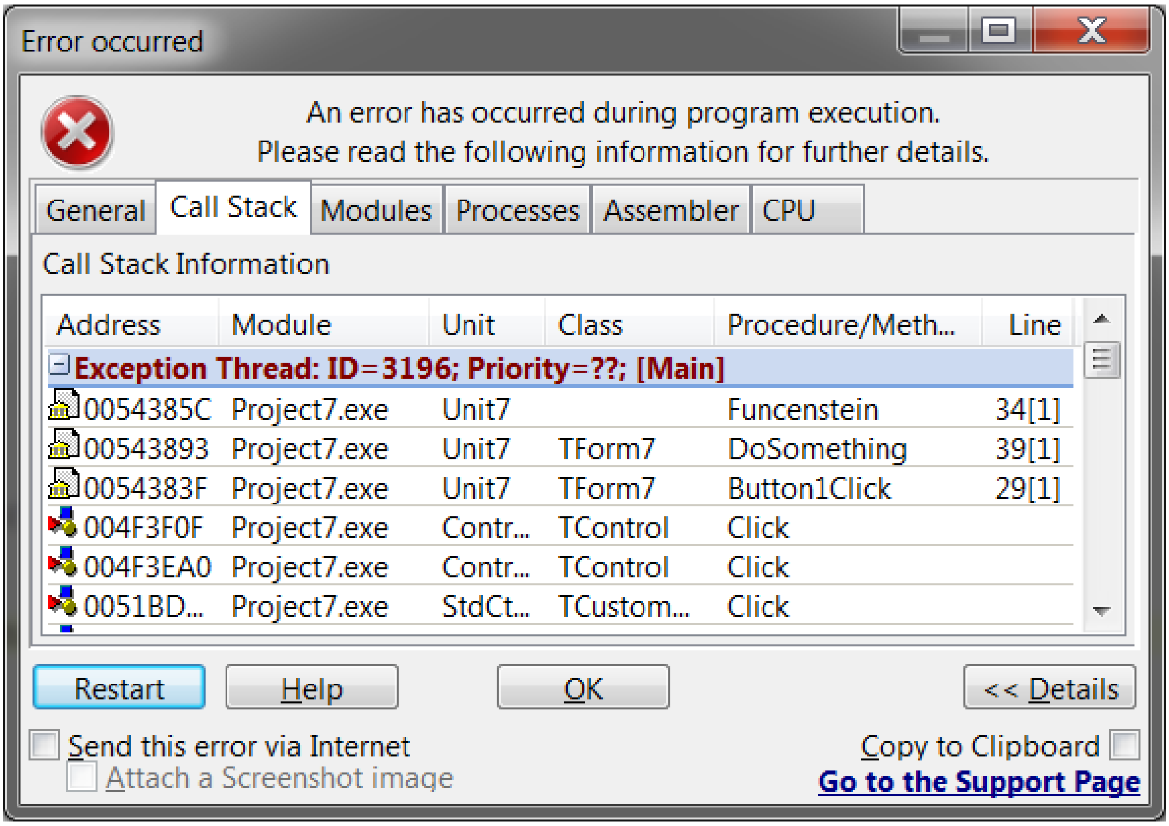


An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
Exception Thread: ID=6404; Priority=??; [Main]					
0054385C	Project7.exe	Unit7		Funcenstein	34[1]
00543893	Project7.exe	Unit7	TForm7	DoSomething	39[1]
0054383F	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F3F0F	Project7.exe	Contr...	TControl	Click	
004F3EA0	Project7.exe	Contr...	TControl	Click	
0051BD...	Project7.exe	StdCt...	TCustom...	Click	

EurekaLog dialog in detailed mode with all options unchecked



Error occurred

An error has occurred during program execution.
Please read the following information for further details.

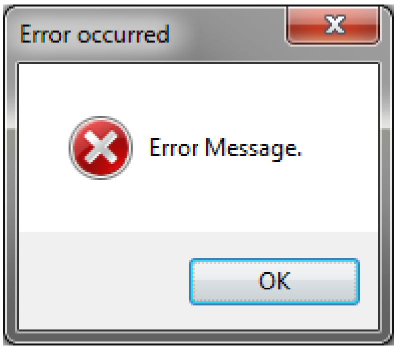
Call Stack Information

Address	Module	Unit	Class	Procedure/Meth...	Line
Exception Thread: ID=3196; Priority=??; [Main]					
0054385C	Project7.exe	Unit7		Funcenstein	34[1]
00543893	Project7.exe	Unit7	TForm7	DoSomething	39[1]
0054383F	Project7.exe	Unit7	TForm7	Button1Click	29[1]
004F3F0F	Project7.exe	Contr...	TControl	Click	
004F3EA0	Project7.exe	Contr...	TControl	Click	
0051BD...	Project7.exe	StdCt...	TCustom...	Click	

☐ Send this error via Internet
☐ Attach a Screenshot image
Copy to Clipboard ☐
Go to the Support Page

EurekaLog dialog in detailed mode with all options checked

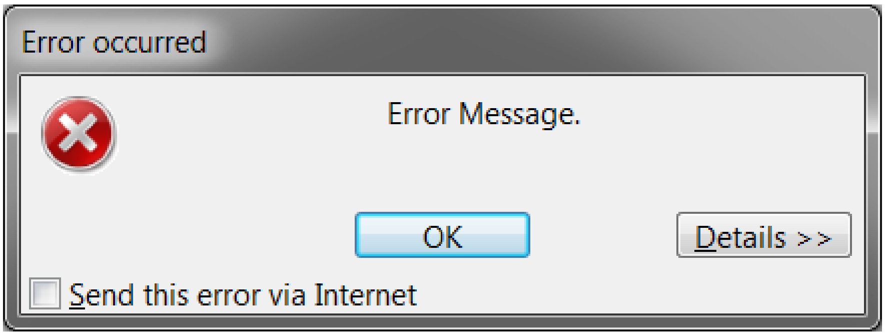
Compare it with typical MessageBox dialog 373:



Error occurred

Error Message.

A typical MessageBox dialog



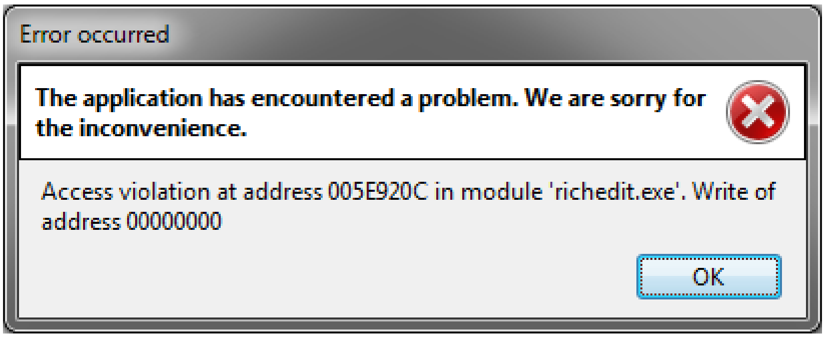
Error occurred

Error Message.

☐ Send this error via Internet

A typical EurekaLog dialog

And here is a comparison with MS Classic dialog 377:

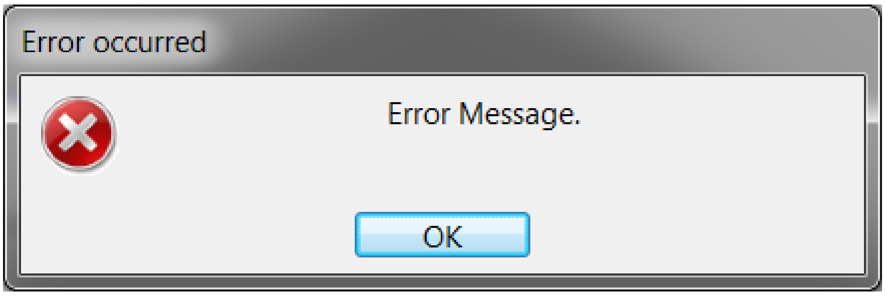


Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Access violation at address 005E920C in module 'richedit.exe'. Write of address 00000000

MS Classic dialog type



Error occurred

Error Message.

EurekaLog dialog type

Constant: edtEurekaLog, edtEurekaLogDetailed.

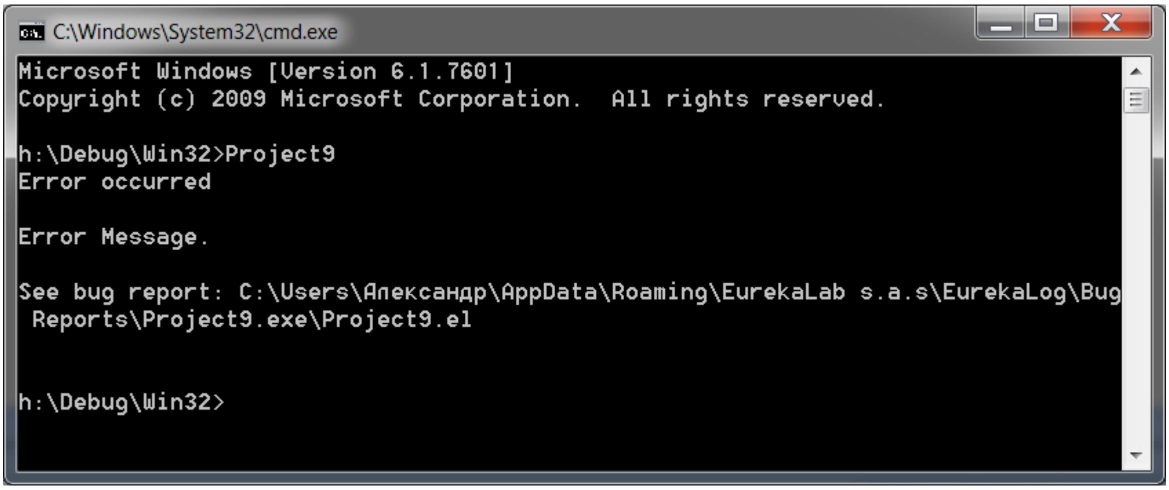
See also:

- EurekaLog dialog options 279
- MS Classic dialog 377
- MessageBox dialog 373
- Other dialogs 370
- Nested (chained) exceptions 244
- Bug reports 72

10.5.6 Console

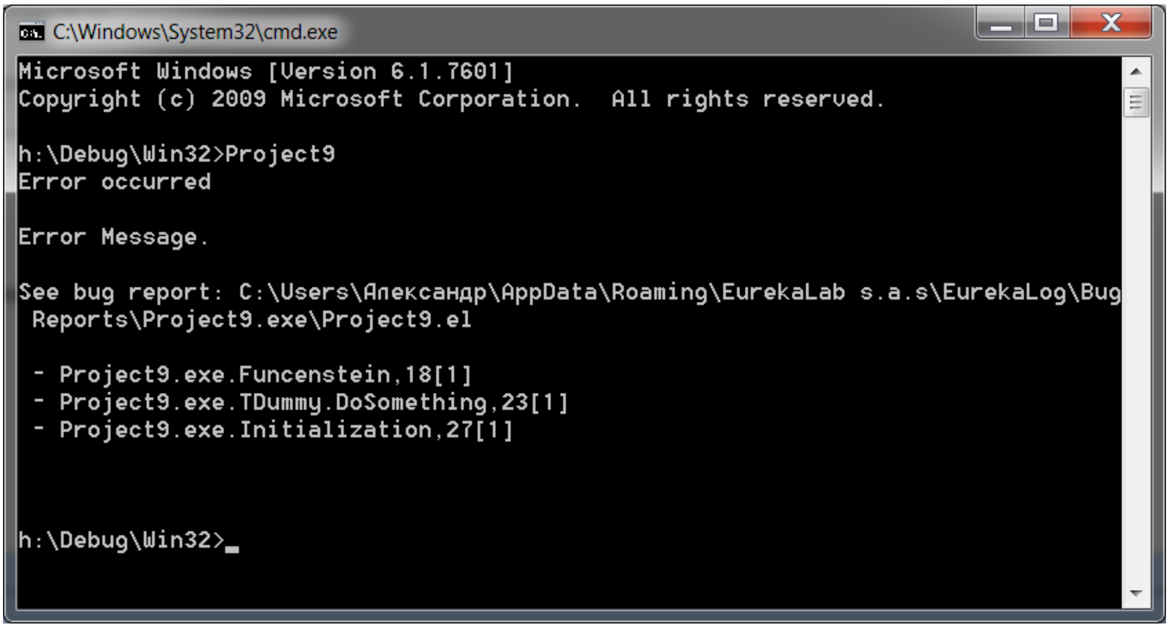
Console dialog is dialog which can only be used in console-based application. This dialog outputs error message to STD_ERROR_HANDLE (by default it's the same as STD_OUTPUT_HANDLE).

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.



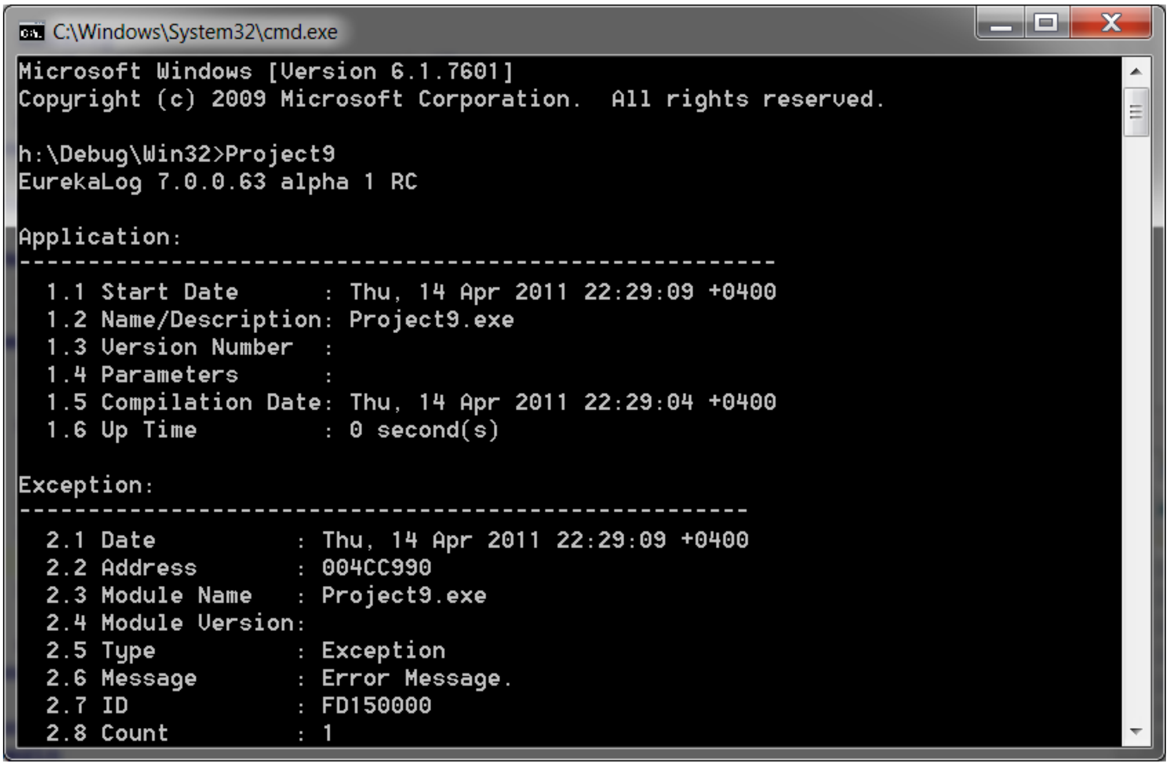
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

See bug report: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Bug
 Reports\Project9.exe\Project9.el


h:\Debug\Win32>
```

Standard console dialog



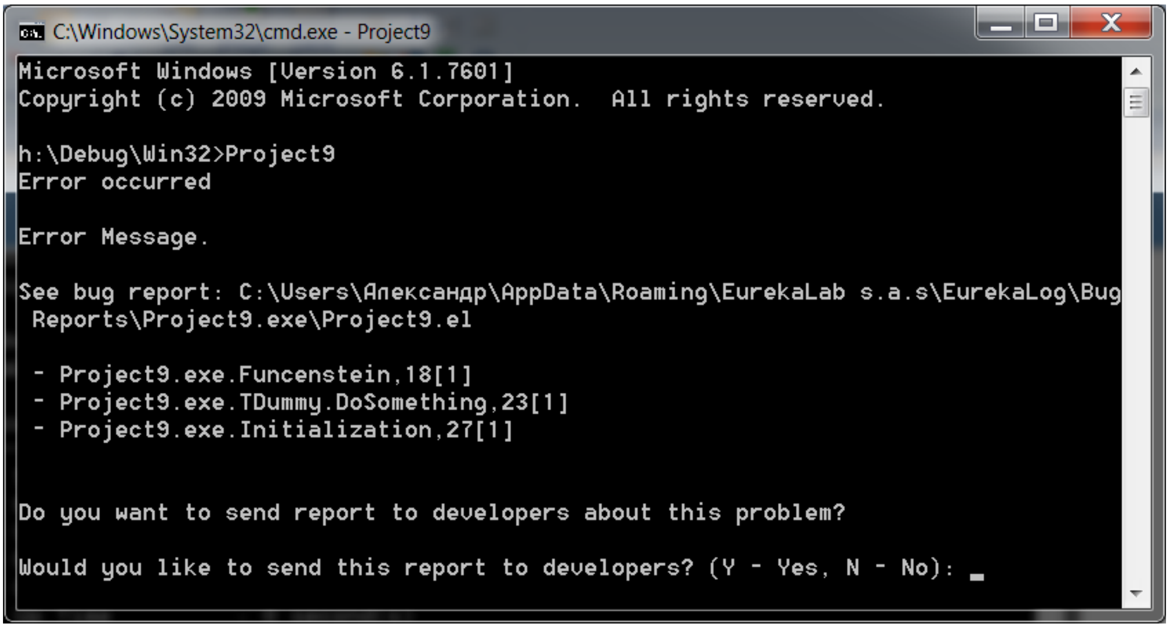
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

See bug report: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Bug
 Reports\Project9.exe\Project9.el

 - Project9.exe.Funcenstein,18[1]
 - Project9.exe.TDummy.DoSomething,23[1]
 - Project9.exe.Initialization,27[1]



h:\Debug\Win32>_
```

Console dialog in detailed mode



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
EurekaLog 7.0.0.63 alpha 1 RC

Application:
-------------------------------------------------------
  1.1 Start Date      : Thu, 14 Apr 2011 22:29:09 +0400
  1.2 Name/Description: Project9.exe
  1.3 Version Number  :
  1.4 Parameters      :
  1.5 Compilation Date: Thu, 14 Apr 2011 22:29:04 +0400
  1.6 Up Time         : 0 second(s)

Exception:
-----------------------------------------------------
  2.1 Date          : Thu, 14 Apr 2011 22:29:09 +0400
  2.2 Address       : 004CC990
  2.3 Module Name   : Project9.exe
  2.4 Module Version:
  2.5 Type          : Exception
  2.6 Message       : Error Message.
  2.7 ID            : FD150000
  2.8 Count         : 1
```

Console dialog in "dump report" mode



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

h:\Debug\Win32>Project9
Error occurred

Error Message.

See bug report: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Bug
 Reports\Project9.exe\Project9.el

 - Project9.exe.Funcenstein,18[1]
 - Project9.exe.TDummy.DoSomething,23[1]
 - Project9.exe.Initialization,27[1]


Do you want to send report to developers about this problem?

Would you like to send this report to developers? (Y - Yes, N - No): _
```

Console dialog in detailed mode and asking for send consent

Note: what's the difference between console dialog type and message box dialog type with "native" mode 373? The message box dialog tries to be as much similar to popup message box window as possible. I.e. it displays the same information and it asks for confirmation - pressing "Enter" in console is like clicking on "OK" button in the message box. On the other hand, console dialog type doesn't try to be similar to anyone. It displays its own set of information and it doesn't wait for user, until it needs user's answer.

"Dump report" mode shows the entire bug report. See this article 72 for more information about bug reports.

Constants: edtConsole, edtConsoleDetailed, edtConsoleDump.

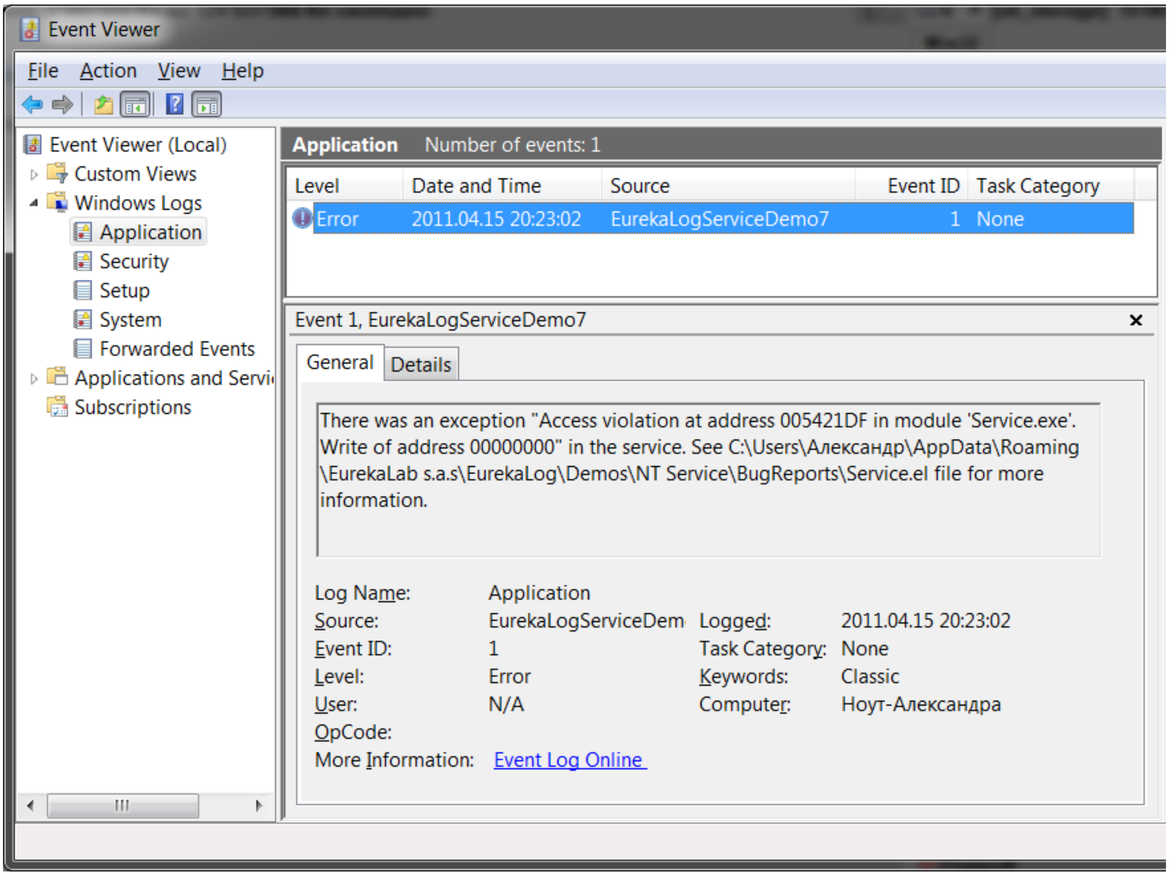
See also:

- Console dialog options 292
- MessageBox dialog 373
- Other dialogs 370
- Nested (chained) exceptions 244
- Bug reports 72

10.5.7 System log

System log dialog behaves like no dialog 371, except it additionally records exception to system event log 535. This dialog type is usually used in service-like applications (like Win32 services, web-applications and so on).

This dialog type has no visual appearance in your application. Here is standard system event viewer with one record from system log dialog:



Level	Date and Time	Source	Event ID	Task Category
Error	2011.04.15 20:23:02	EurekaLogServiceDemo7	1	None

Event 1, EurekaLogServiceDemo7

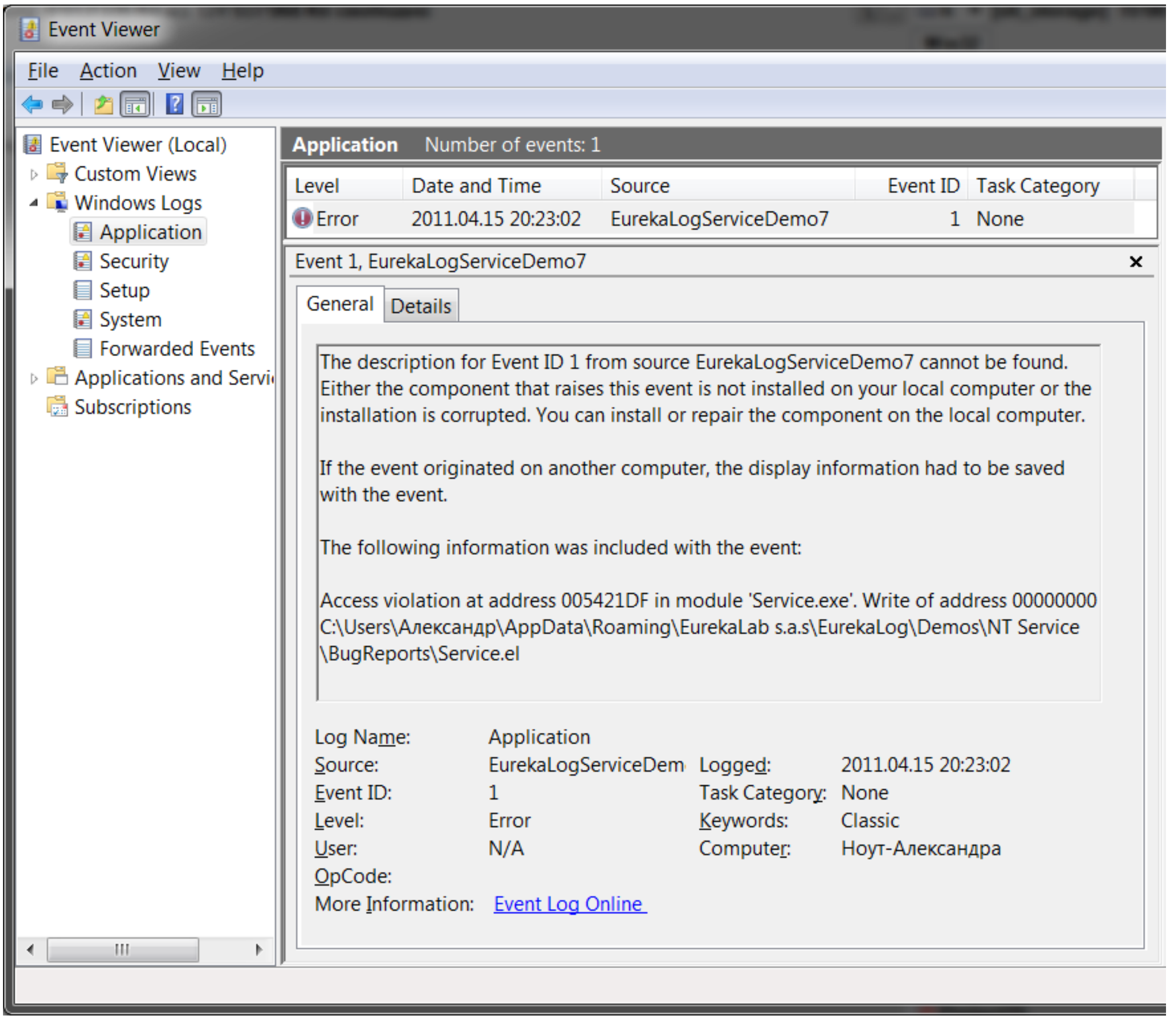
General Details

There was an exception "Access violation at address 005421DF in module 'Service.exe'. Write of address 00000000" in the service. See C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Demos\NT Service\BugReports\Service.el file for more information.

Log Name:	Application		
Source:	EurekaLogServiceDem	Logged:	2011.04.15 20:23:02
Event ID:	1	Task Category:	None
Level:	Error	Keywords:	Classic
User:	N/A	Computer:	Ноут-Александра
OpCode:			
More Information:	Event Log Online		

System event log with new event about exception in EurekaLog-enabled application

You can't just switch your application to this dialog - it requires additional setup steps 535: your application must register itself in the system event log 537 in order for this to work properly. If you fail to perform a proper registering - then event log will be displayed incorrectly for your application, for example:



Level	Date and Time	Source	Event ID	Task Category
Error	2011.04.15 20:23:02	EurekaLogServiceDemo7	1	None

The description for Event ID 1 from source EurekaLogServiceDemo7 cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer.

If the event originated on another computer, the display information had to be saved with the event.

The following information was included with the event:

Access violation at address 005421DF in module 'Service.exe'. Write of address 00000000
C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\Demos\NT Service\BugReports\Service.el

Log Name: Application
Source: EurekaLogServiceDem Logged: 2011.04.15 20:23:02
Event ID: 1 Task Category: None
Level: Error Keywords: Classic
User: N/A Computer: Ноут-Александра
OpCode:
More Information: Event Log Online

The same event in system log, but without proper application's registration

Please, refer to system logging setup 535 article to know more about your application's registration.
Please, refer to system log dialog options 543 article to know more about configuring the dialog itself.

Constant: edtService.

See also:

- System logging setup 535
- System log dialog options 295
- (none) dialog 371
- Other dialogs 370

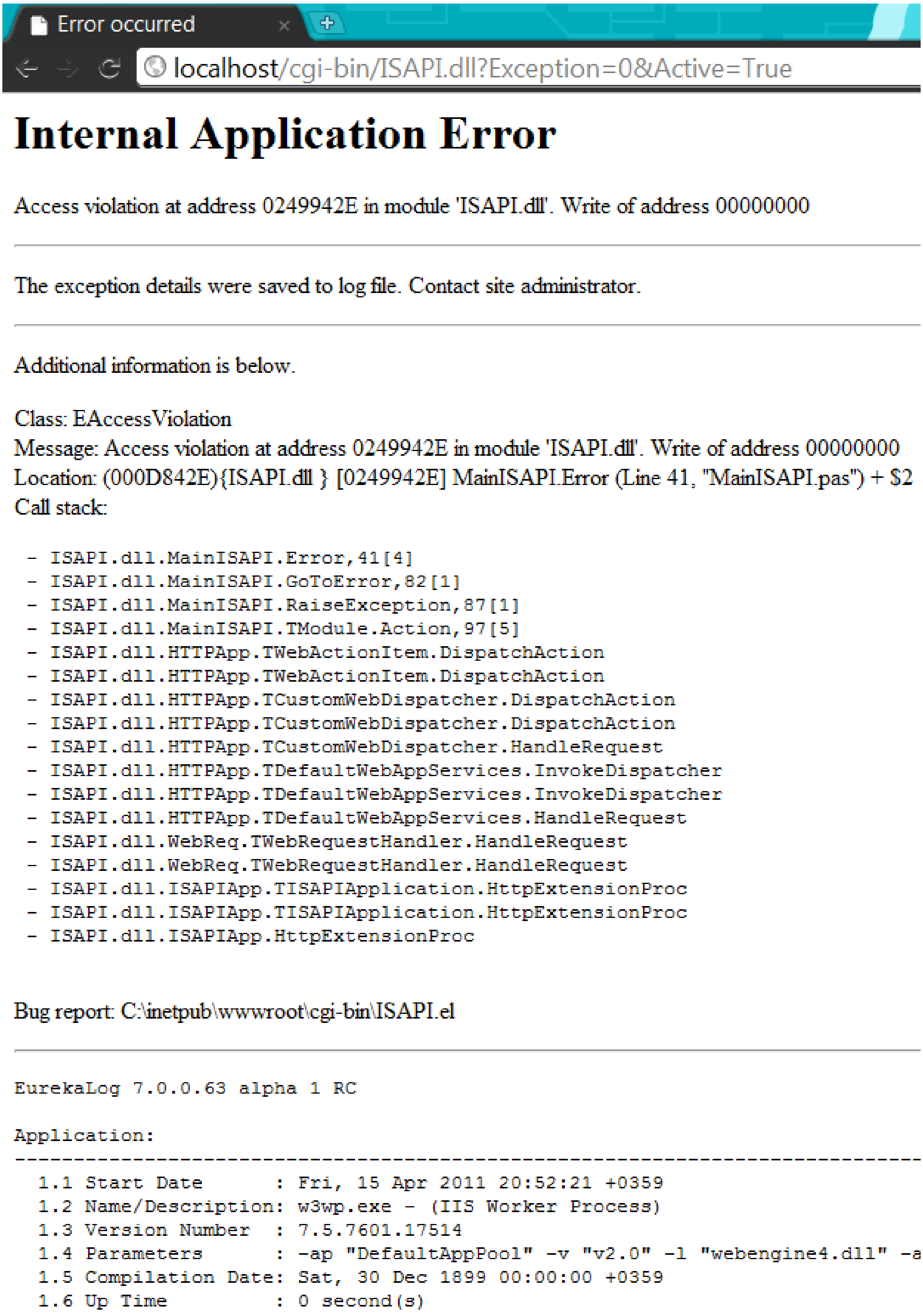
10.5.8 WEB

WEB dialog is a special dialog which behaves like no dialog 371, except it also sends HTML page with error message back to client in web applications. Web application include CGI applications (both console and WinCGI), ISAPI applications, IntraWeb applications or any other HTTPApp-based application.

Warning: if there will be error during stage other than processing client's request (like: application's initialization or shutdown) - there will be no HTML page, since there is no client connected. So be sure that you've enabled and setup bug report saving 46. **Be sure that bug report folder is writable under user account, which is used to run your application**.

Note: error messages in dialogs are controlled by nested exceptions behaviour options 244.

This dialog type has no visual appearance in your application. Here is a browser with error message from your web-application:



Internal Application Error

Access violation at address 0249942E in module 'ISAPI.dll'. Write of address 00000000

The exception details were saved to log file. Contact site administrator.

Additional information is below.

Class: EAccessViolation
Message: Access violation at address 0249942E in module 'ISAPI.dll'. Write of address 00000000
Location: (000D842E){ISAPI.dll } [0249942E] MainISAPI.Error (Line 41, "MainISAPI.pas") + $2
Call stack:

```
 - ISAPI.dll.MainISAPI.Error,41[4]
 - ISAPI.dll.MainISAPI.GoToError,82[1]
 - ISAPI.dll.MainISAPI.RaiseException,87[1]
 - ISAPI.dll.MainISAPI.TModule.Action,97[5]
 - ISAPI.dll.HTTPApp.TWebActionItem.DispatchAction
 - ISAPI.dll.HTTPApp.TWebActionItem.DispatchAction
 - ISAPI.dll.HTTPApp.TCustomWebDispatcher.DispatchAction
 - ISAPI.dll.HTTPApp.TCustomWebDispatcher.DispatchAction
 - ISAPI.dll.HTTPApp.TCustomWebDispatcher.HandleRequest
 - ISAPI.dll.HTTPApp.TDefaultWebAppServices.InvokeDispatcher
 - ISAPI.dll.HTTPApp.TDefaultWebAppServices.InvokeDispatcher
 - ISAPI.dll.HTTPApp.TDefaultWebAppServices.HandleRequest
 - ISAPI.dll.WebReq.TWebRequestHandler.HandleRequest
 - ISAPI.dll.WebReq.TWebRequestHandler.HandleRequest
 - ISAPI.dll.ISAPIApp.TISAPIApplication.HttpExtensionProc
 - ISAPI.dll.ISAPIApp.TISAPIApplication.HttpExtensionProc
 - ISAPI.dll.ISAPIApp.HttpExtensionProc
```

Bug report: C:\inetpub\wwwroot\cgi-bin\ISAPI.el

```
EurekaLog 7.0.0.63 alpha 1 RC

Application:
-----------------------------------------------------------------------------
  1.1 Start Date      : Fri, 15 Apr 2011 20:52:21 +0359
  1.2 Name/Description: w3wp.exe - (IIS Worker Process)
  1.3 Version Number  : 7.5.7601.17514
  1.4 Parameters      : -ap "DefaultAppPool" -v "v2.0" -l "webengine4.dll" -a
  1.5 Compilation Date: Sat, 30 Dec 1899 00:00:00 +0359
  1.6 Up Time         : 0 second(s)
```

Google Chrome shows error page about exception which occurred during HTTP-request

The example error message above uses the following HTML page template:

```
<html>
  <head>
    <%content_type%>
    <title>EurekaLog 7 Documentation</title>
  </head>
  <body>
    <h1>Internal Application Error</h1>
    <p><%EXCEPTION_MESSAGE%></p>
    <p><hr width="100%"></p>
    <p>The exception details were saved to log file. Contact site administrator.</p>
    <p><hr width="100%"></p>
    <p>Additional information is below.</p>
    <p>Class: <%exception_class%><br />
       Message: <%exception_message%><br />
       Location: <%exception_location%><br />
       Call stack: <%call_stack%><br />
       Bug report: <%file_name%></p>
    <p><hr width="100%"></p>
    <p><%bug_report%></p>
  </body>
</html>
```

The HTML source of the result looks like this:

```
<html>
  <head>
    <meta http-equiv="content-type" content="TEXT/HTML;charset=UTF-8" />
    <title>Error occurred</title>
  </head>
  <body>
    <h1>Internal Application Error</h1>
    <p>Access violation at address 0216942E in module 'ISAPI.dll'. Write of address 0000000
    <p><hr width="100%"></p>
    <p>The exception details were saved to log file. Contact site administrator.</p>
    <p><hr width="100%"></p>
    <p>Additional information is below.</p>
    <p>Class: EAccessViolation<br />
       Message: Access violation at address 0216942E in module 'ISAPI.dll'. Write of addres
       Location: (000D842E){ISAPI.dll   } [0216942E] MainISAPI.Error (Line 41, "MainISAPI.p
       Call stack: <pre>
 - ISAPI.dll.MainISAPI.Error,41[4]
 ...
 - ISAPI.dll.ISAPIApp.HttpExtensionProc
</pre>
<br />
       Bug report: C:\inetpub\wwwroot\cgi-bin\ISAPI.el</p>
    <p><hr width="100%"></p>
    <p><pre>EurekaLog 7.0.0.63 alpha 1 RC
...
0157ECF0: 0157ED0C   0216951E: 95 16 02 E8 92 43 F5 FF 8B 45 E8 B9 4C 00 00 00  .....C...E.
</pre></p>
  </body>
</html>
<!-- EurekaLog page ID: CC2F96D8 -->
```

```
<!-- EurekaLog Bug ID: 7D390000 -->
```

Note: it may be a good idea to hide information about occurred problem in application's release version, showing only generic message "`The exception details were saved to log file. Contact site administrator`" and logging bug report to file. Showing too much error details may compromise your security.

Constant: edtWeb.

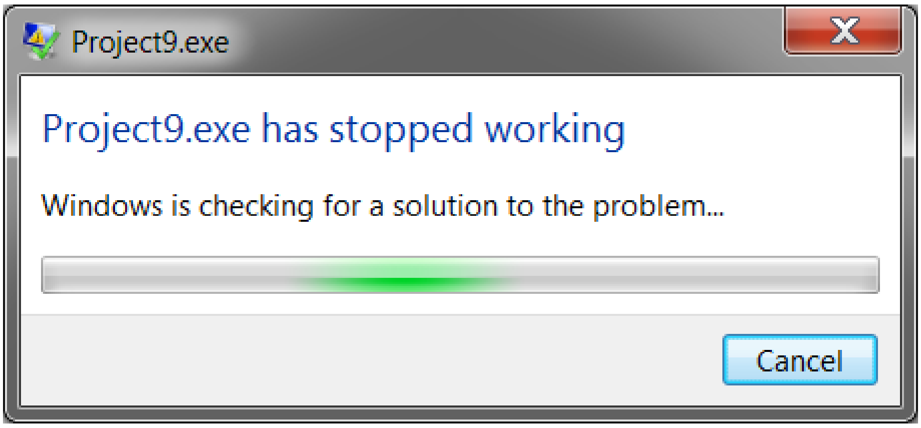
See also:

- WEB dialog setup 296
- (none) dialog 371
- Other dialogs 370
- Nested (chained) exceptions 244

10.5.9 WER

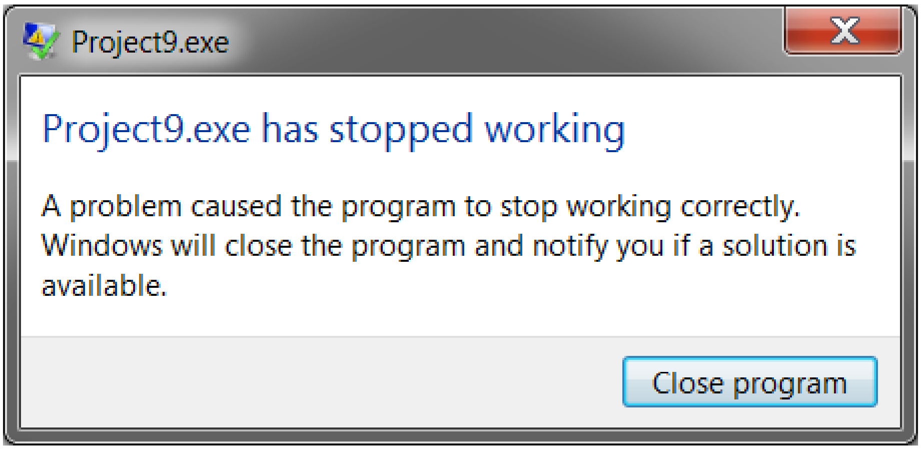
WER (Windows Error Reporting) dialog is a standard system error dialog. This dialog is preferred dialog in your application, if you can afford it (see using WER 573 for more info). Otherwise you can use the MS Classic dialog 377 instead.

Visual appearance and behaviour of this dialog depends on host OS and host OS's settings. It doesn't depend on your application, though you have some options to change (see customization 300).



Project9.exe has stopped working

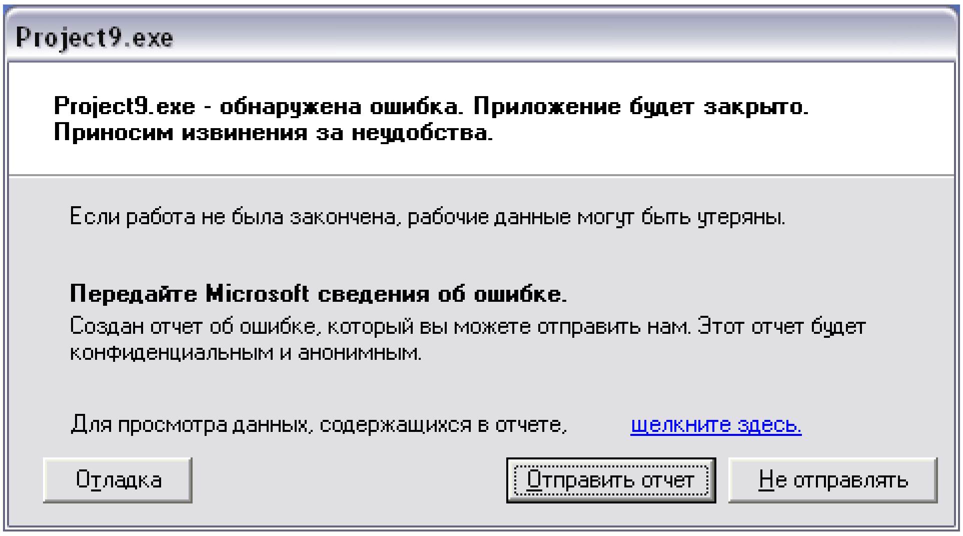
Windows is checking for a solution to the problem...



Project9.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

Windows Error Reporting on Windows 7



Project9.exe

Project9.exe - обнаружена ошибка. Приложение будет закрыто. Приносим извинения за неудобства.

Если работа не была закончена, рабочие данные могут быть утеряны.

Передайте Microsoft сведения об ошибке.

Создан отчет об ошибке, который вы можете отправить нам. Этот отчет будет конфиденциальным и анонимным.

Для просмотра данных, содержащихся в отчете, щелкните здесь.

Windows Error Reporting on Windows XP

This dialog invokes system error processing. It works on any supported OS 10, but it's highly dependent on host OS's capabilities.

Notes:

1. This dialog sends bug report to Microsoft's server using the standard procedure. Any sending method that you setup in EurekaLog's options will be ignored. However, you still can send your custom files (this feature may be not supported on all OSs).
2. The capabilities of this dialog depends on host OS. Not all options are available for all OSs. Please, refer to MSDN articles.
3. Usually it's not a good idea to use this kind of dialog for reporting leaks.
4. Your application will be terminated after this dialog, so be sure to setup some restart and recovery options.

Constant: edtWER.

See also:

- WER dialog options 300
- Using WER 573
- RTL dialog 371
- MS Classic dialog 377
- Other dialogs 370
- MSDN: Windows Error Reporting

10.6 Send engines

You can setup sending report to developer (you) via one or several available methods 53.

There are many send methods available:

Send method	Type	Based on	Description
Shell 391	e-mail	SMTP	Send e-mail via ShellExecute function (mailto: protocol).
Simple MAPI 393	e-mail	SMTP	Send e-mail via Simple MAPI protocol.
MAPI 396	e-mail	SMTP	Send e-mail via MAPI protocol.
SMTP client 397	e-mail	SMTP	Send e-mail via SMTP client (similar to usual e-mail client software).

SMTP server 398	e-mail	SMTP	Send e-mail via fake SMTP server.
HTTP upload 398	WEB	HTTP + any scripting lang, CGI or ISAPI	Send bug report file via script on HTTP server.
FTP upload 404	WEB	FTP	Direct bug report file's upload on FTP server.
FogBugz 404	Bug tracker	HTTP + ASP.NET (Win)/PHP (*nix) + XML	Send bug report to FogBugz bug tracker.
Mantis 406	Bug tracker	HTTP + PHP + SOAP	Send bug report to Mantis bug tracker.
BugZilla 407	Bug tracker	HTTP + Perl + XML-RPC	Send bug report to BugZilla bug tracker.
JIRA 408	Bug tracker	HTTP + JAVA + JSON	Send bug report to JIRA bug tracker.

You can know more about each send method by clicking on each method in the table above.

See also:
- Selecting best send method 55
- Comparison of issue-tracking systems
- Security Considerations 158
- Configuring sending 53
- Configuring send options 302
- Using unsupported bug tracker software 153

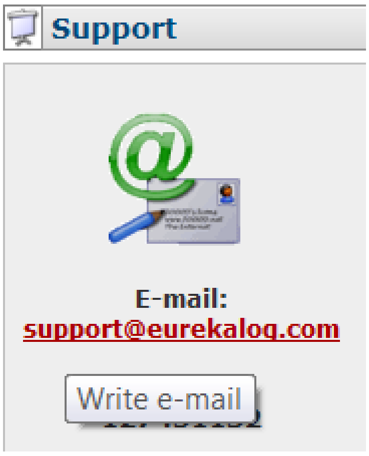
10.6.1 Shell

This method sends bug report via `ShellExecute(..., 'mailto:example@example.com?subject=error&body=report', ...)`.

The mailto URI scheme, as registered with the Internet Assigned Numbers Authority (IANA), defines the scheme for Simple Mail Transfer Protocol (SMTP) email addresses. Though its use is not strictly defined, URLs of this form are intended to be used to open the new message window of the user's email client when the URL is activated, with the address as defined by the URL in the "To:" field.

The software mechanism activated by the link requires that a default email client be established on the computer. This must be a local program, typically using the SMTP protocol to send outbound mail. With the rise in use of webmail-based email, many computers lack local email client software. Alternatively, email client software may have been preinstalled by the computer vendor, but never used or configured.

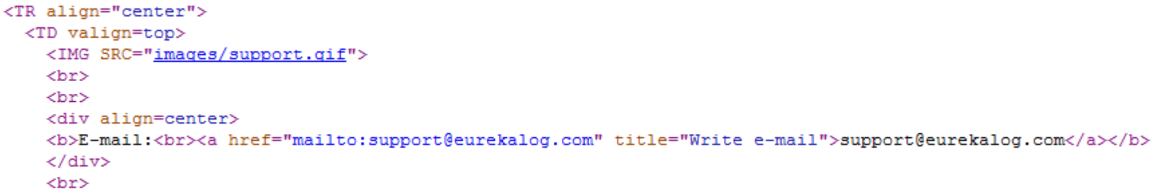
This is the same protocol, which is used to send mails from web site in browser. For example:



Support

E-mail:
support@eurekalog.com

mailto link in web-browser



```
<TR align="center">
  <TD valign=top>
    <IMG SRC="images/support.gif">
    <br>
    <br>
    <div align=center>
    <b>E-mail:<br><a href="mailto:support@eurekalog.com" title="Write e-mail">support@eurekalog.com</a></b>
    </div>
    <br>
```

HTML source code for example above

Examples of e-mail clients with mailto protocol support: Windows Mail, Outlook Express, Outlook, Mozilla Thunderbird, The Bat!. Well, most e-mail clients support it.

Advantages:

- Available (almost) always, since it's very easy to implement and it's included in basic operation system configuration.
- Have very high chances to succeed, since it uses client's configuration to send e-mail and client most likely have e-mail client installed and configured (and if not - client still can save bug report to file and send it via other way).
- User uses his real e-mail address, so you can always contact him for more info.
- Most common e-mail programs support mailto protocol.
- Good for basic support for unsupported web-trackers (see also 153).

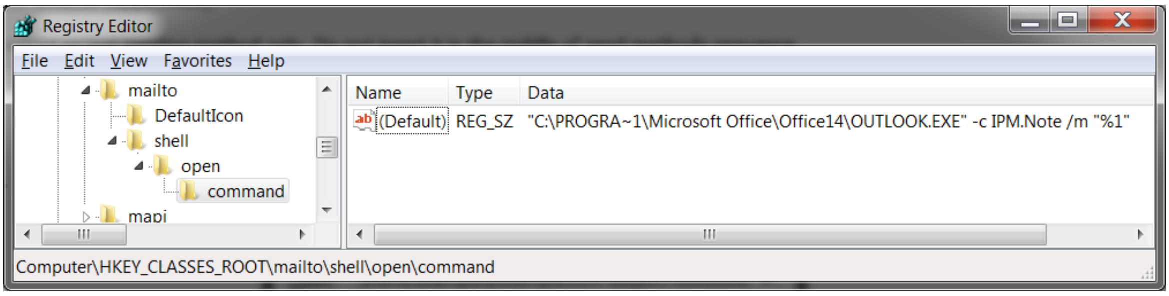
Drawbacks:

- Depends on client's environment. You can't control it.
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management.
- UI interaction: requires user to click on "Send" in their e-mail clients. Automatic send without user actions is not possible.
- Can be canceled.
- Not customizable at all.
- Limitation on message size (command-line size limitation).
- Unable to send any files - message text only.
- Unicode is rarely supported.
- Poor (non-strict) implementation of protocol may result in encoding problems.
- Always "succeed", no way to get real send result.
- Launched in separate window (application).
- SSL/TLS support may be not present.

Warning: Use this method as last sending method only. Do not insert it in the middle of send methods sequence.

E-mail client is registered under HKEY_CLASSES_ROOT\mailto registry key (HKEY_CURRENT_USER\SOFTWARE\Classes\mailto or HKEY_LOCAL_MACHINE\SOFTWARE\Classes

\mailto):



(Default) REG_SZ "C:\PROGRA~1\Microsoft Office\Office14\OUTLOOK.EXE" -c IPM.Note /m "%1"

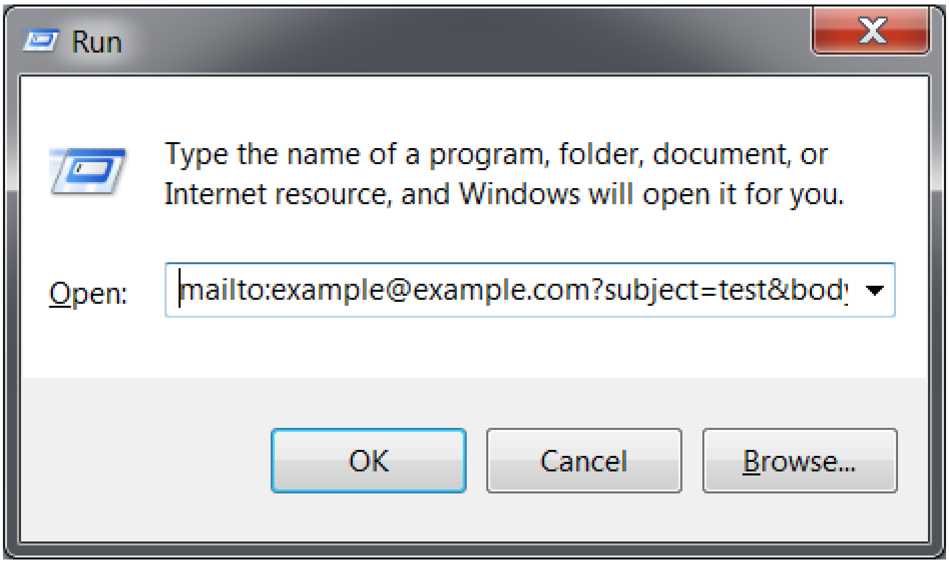
Computer\HKEY_CLASSES_ROOT\mailto\shell\open\command

Outlook 2010 is registered as the handler for mailto protocol

Registration as the default Start menu email application is not equivalent to registration as the system default email client or the registered mailto handler:

- The system default email client is started when the user clicks "Read e-mail" from the Internet Explorer Tools menu.
- The registered mailto handler is started when the user clicks a URL of the form `mailto:someone@example.com`.
- The Start menu email application is started when the user clicks the e-mail icon on the Start menu.
- If no default Start menu email application is specified, the e-mail icon on the Start menu launches the system default email client.

You can test it manually by typing "`mailto:example@example.com?subject=test&body=test`" (without quotes) in Start/Run dialog box:


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

mailto:example@example.com?subject=test&bod

Start / Run dialog with "mailto:" URL

Constant: esmShellClient.

See also:

- Shell method options 309
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

10.6.2 Simple MAPI

Simple MAPI is a subset of 12 functions (compared to MAPI 396), which enable developers to add basic messaging functionality to their Windows-based applications. Simple MAPI includes functions to support sending and receiving messages:

- Log onto the messaging system.
- Compose new messages, add and resolve recipients, send messages.
- Retrieve and read messages from the inbox.

The Simple MAPI functions can be called from any application that supports both making API calls as well as the structures and data-types used by Simple MAPI, such as Delphi, C, C++, Visual Basic, and Visual Basic for Applications (VBA).

For more information specific to Simple MAPI see the following KnowledgeBase articles:

- 105964 PC MAPI: Simple MAPI Common Technical Questions and Answer (**FAQ**)
- 239576 INFO: Developer Support Limitations with Outlook Express

Examples of e-mail clients with Simple MAPI protocol support: Windows Mail, Outlook Express, Outlook (no longer supported since Outlook 2007), Mozilla Thunderbird, The Bat!

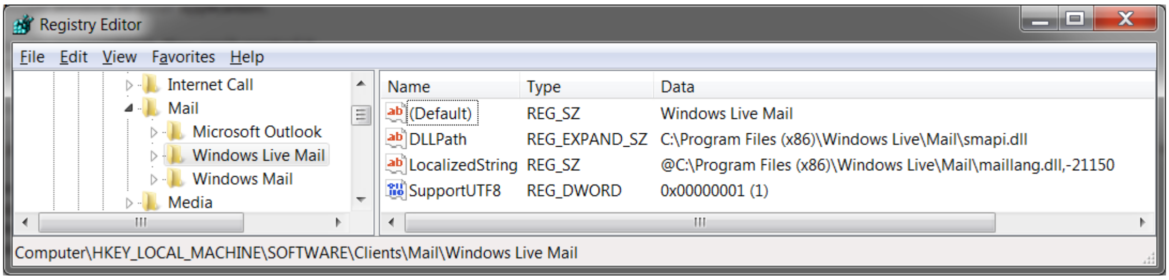
Advantages:

- It's simple protocol, which is relatively easy to implement (supported by most e-mail clients).
- Most common e-mail programs support Simple MAPI protocol.
- Have good chances to succeed, since it uses client's configuration to send e-mail and client most likely have e-mail client installed and configured (and if not - client still can save bug report to file and send it via other way).
- User uses his real e-mail address, so you can always contact him for more info.
- Launched as modal window in your application.
- Good for basic support for unsupported web-trackers (see also 153).
- No problems with 32<->64 interoperability.

Drawbacks:

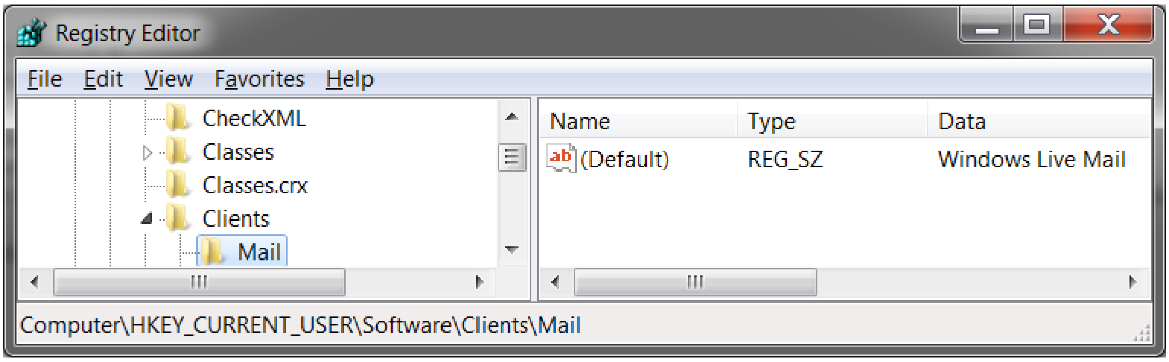
- Depends on client's environment. You can't control it.
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management.
- Obsolete protocol, which may be unavailable in the next versions of operating system (modern versions of Outlook do not support it).
- UI interaction: requires user to click on "Send" in their e-mail clients. Automatic send without user actions is not possible.
- Even if user has e-mail client installed and configured - this e-mail client software still may not implement Simple MAPI protocol.
- Little customization possibilities.
- No unicode support.
- Can be canceled.
- May be confusing for user (* 396).
- SSL/TLS support may be not present.

E-mail client register itself with Simple MAPI protocol by creating a sub-key in HKEY_LOCAL_MACHINE\Software\Clients\Mail\ registry key and setting default value for `HKEY_LOCAL_MACHINE\Software\Clients\Mail\` to your sub-key name (use HKEY_CURRENT_USER for local user only). For example:



Name	Type	Data
(Default)	REG_SZ	Windows Live Mail
DLLPath	REG_EXPAND_SZ	C:\Program Files (x86)\Windows Live\Mail\smapi.dll
LocalizedString	REG_SZ	@C:\Program Files (x86)\Windows Live\Mail\maillang.dll,-21150
SupportUTF8	REG_DWORD	0x00000001 (1)

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Clients\Mail\Windows Live Mail



Name	Type	Data
(Default)	REG_SZ	Windows Live Mail

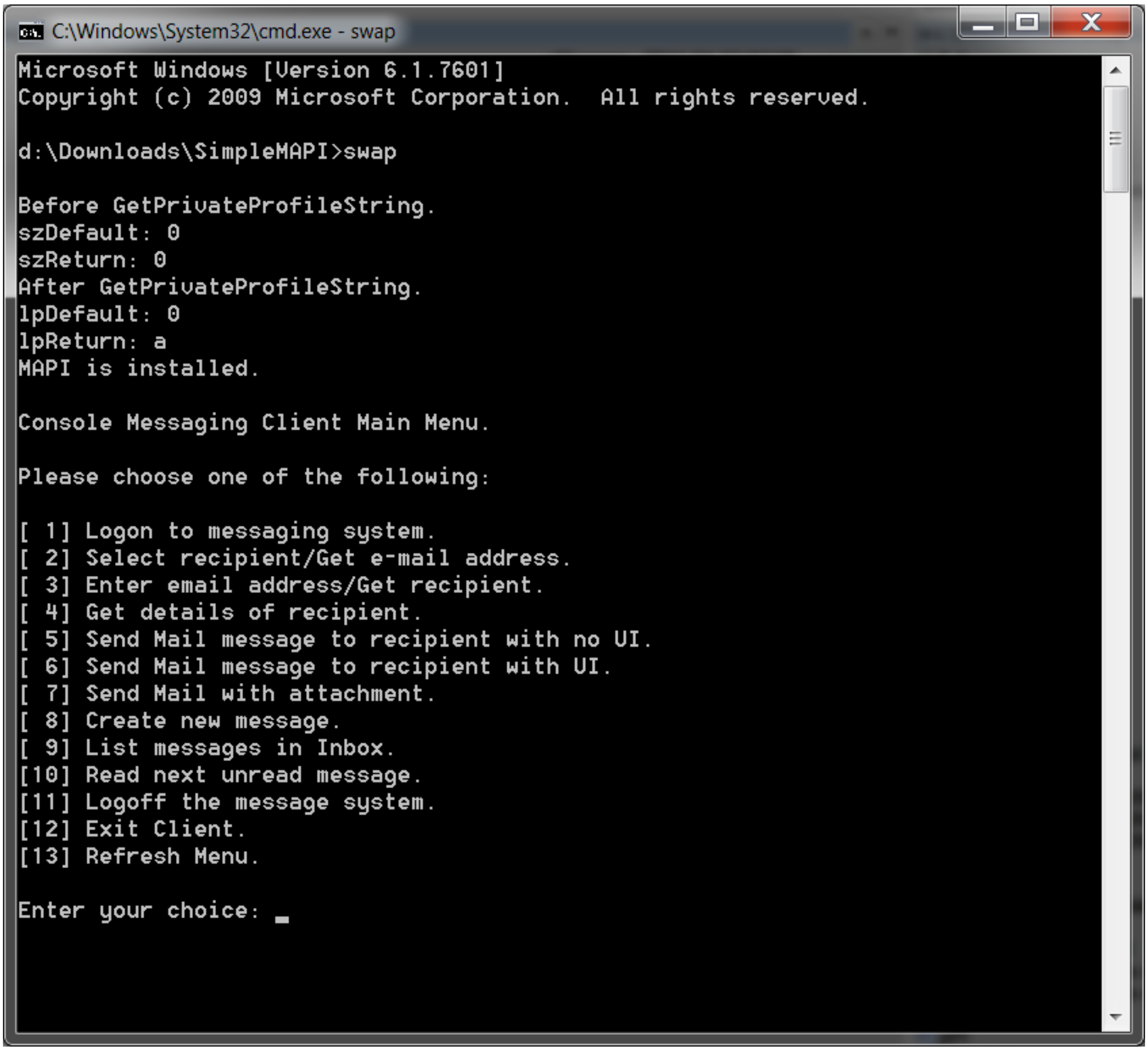
Computer\HKEY_CURRENT_USER\Software\Clients\Mail

Windows Live Mail is registered as simple MAPI client

`DLLPath` value specifies DLL for simple MAPI, `DLLPathEx` value specifies DLL for (extended) MAPI.

Simple MAPI client loads `MAPI32.dll` library from `System` folder. This is a MAPI stub, which reads the above mentioned registry settings, loads proper simple MAPI DLL and redirects all calls to it.

You can test it manually by using Simple MAPI console test application from Microsoft:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

d:\Downloads\SimpleMAPI>swap

Before GetPrivateProfileString.
szDefault: 0
szReturn: 0
After GetPrivateProfileString.
lpDefault: 0
lpReturn: a
MAPI is installed.

Console Messaging Client Main Menu.

Please choose one of the following:

[ 1] Logon to messaging system.
[ 2] Select recipient/Get e-mail address.
[ 3] Enter email address/Get recipient.
[ 4] Get details of recipient.
[ 5] Send Mail message to recipient with no UI.
[ 6] Send Mail message to recipient with UI.
[ 7] Send Mail with attachment.
[ 8] Create new message.
[ 9] List messages in Inbox.
[10] Read next unread message.
[11] Logoff the message system.
[12] Exit Client.
[13] Refresh Menu.

Enter your choice: _
```

Interface of Simple MAPI test tool

Remarks:
(*) That's because, if you have two e-mail client installed (say, Windows Mail and Outlook) - both will definitely support mailto protocol, but only one can support simple MAPI, so you may launch non-default e-mail client (which is not configured). For example, if you have Outlook 2010 as your default e-mail client and you use simple MAPI - it will launch Windows Mail client, because Outlook 2010 doesn't support simple MAPI.

Constant: esmSimpleMAPI.

See also:
- Simple MAPI method options 314
- (Extended) MAPI send method 396
- Differences between Simple MAPI and Extended MAPI
- Other send methods 390
- Selecting send method 55
- Security Considerations 158
- Building MAPI Applications on 32-Bit and 64-Bit Platforms

10.6.3 MAPI

MAPI (also known as **Extended MAPI** or **MAPI 1.0**) - The Messaging Application Program Interface - is a messaging architecture and a Component Object Model based API for Microsoft Windows. MAPI allows client programs to become (e-mail) messaging-enabled, -aware, or -based by calling MAPI subsystem routines that interface with certain messaging servers. While MAPI is designed to be independent of the protocol, it is usually used with MAPI/RPC, the proprietary protocol that Microsoft Outlook uses to communicate with Microsoft Exchange.

Simple MAPI 393 is a subset of 12 functions which enable developers to add basic messaging functionality. Extended MAPI allows complete control over the messaging system on the client computer, creation and management of messages, management of the client mailbox, service providers, and so forth. Simple MAPI ships with Microsoft Windows as part of Outlook Express/Windows Mail while the full Extended MAPI ships with Office Outlook and Exchange.

In addition to the Extended MAPI client interface, programming calls can be made indirectly through the Simple MAPI API client interface, through the Common Messaging Calls (CMC) API client interface, or by the object-based CDO Library interface. These three methods are easier to use and designed for less complex messaging-enabled and -aware applications. (Simple MAPI and CMC were removed from Exchange 2003.)

MAPI includes facilities to access message transports, message stores, and directories.

Examples of e-mail clients with MAPI protocol support: Outlook, Exchange.

Advantages:
- Have good chances to succeed, if e-mail client supports this protocol - since it uses client's configuration to send e-mail.
- User uses his real e-mail address, so you can always contact him for more info.
- Can deliver reports automatically. No UI.
- Good for basic support for unsupported web-trackers (see also 153).

Drawbacks:
- Depends on client's environment. You can't control it.
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management.
- This is complex protocol. Many things can go wrong.
- There can be problems with x32 <-> x64 interoperability, since 3rd party DLL must be loaded in your process.
- This protocol is implemented **very** rarely (MAPI is used only by Outlook and Exchange).
- Can be canceled.
- May be confusing for user (* 397).

- SSL/TLS support may be not present.

MAPI client loads `MAPI32.dll` library from `System` folder. This is a MAPI stub, which reads the registry settings, loads proper MAPI DLL and redirects all calls to it.

You can test MAPI by using OutlookSpy and MFCMapi tools.

Remarks:
(*) That's because, if you have two e-mail client installed (say, Windows Mail and Outlook) - both will definitely support mailto protocol, but only one can support MAPI, so you may launch non-default e-mail client (which is not configured). For example, if you have Windows Mail as your default e-mail client and you use MAPI - it will launch Outlook 2010 client, because Windows Mail doesn't support MAPI.

Constant: esmMAPI.

See also:
- MAPI method options 315
- Simple MAPI send method 393
- Differences between Simple MAPI and Extended MAPI
- Other send methods 390
- Selecting send method 55
- Security Considerations 158
- Building MAPI Applications on 32-Bit and 64-Bit Platforms

10.6.4 SMTP client

SMTP client method is just a usual e-mail client software inside your application - the only differences is that it's "send only" (no receive) and very simple. As any other real e-mail client (like Outlook, The Bat!, etc) it requires a real account (e-mail and password).

Advantages:
- No additional client software or configuration necessary.
- Emulates real e-mail software client. **Most reliable e-mail delivery method**.
- Can deliver reports automatically. No UI.
- SSL/TLS support.
- Good for basic support for unsupported web-trackers (see also 153).

Drawbacks:
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management.
- Can be blocked by firewall or client's ISP, since you don't use settings of user's e-mail application (some ISPs require to use their SMTP relay servers to send e-mails outside).
- "FROM" field is (almost) always your account.
- User e-mail is optional.
- **You must store your account details (login/password) in your application**. Real e-mail account is required, since it's actually a real e-mail client inside application.

Warning: your real account's data will be stored inside application. Even if it's encrypted - it's still stored inside .exe, so **it can be stolen**. **DO NOT** use your personal e-mail for this. Create a new special account for bug reporting via this method (and be sure to protect it against e-mail/password change or hi-jacking).

Currently EurekaLog supports AUTH LOGIN and AUTH PLAIN authentication methods.

Typically, you should use either SMTP server 398 or SMTP client, but not both methods simultaneously.

Constant: esmSMTPClient.

See also:

- SMTP client method options 316
- SMTP server send method 398
- Other send methods 390
- Selecting send method 55
- Security Considerations 158
- Differences between SSL and TLS modes 588

10.6.5 SMTP server

SMTP server method creates a temporary e-mail server (like real e-mail server of AOL Mail, GMail, HotMail, Yahoo Mail, etc), except it exists only during sending bug report (i.e. it's fake e-mail server). Since SMTP server stores user accounts in itself (no need to pass any account's data to 3rd party service) - you don't need to create any real e-mail accounts: fake SMTP server can use fake e-mail account.

Once created, fake SMTP server will connect to other (real) SMTP server, which serves target (recipient) account, saying: "Hi, I'm e-mail server and my account (it's fake, but you don't know this) just sent a new email for one of your account, here it is". After transmission, the fake SMTP server will be destroyed, and real SMTP server will notify recipient about new incoming e-mail.

Note: since SMTP server is fake, it has high chances to be recognized as spam (and, thus, being rejected), so be sure to setup bypass filters in your e-mail account, so e-mail server will not use spam filtering on such e-mails. Also, be sure to setup some other backup send method.

Advantages:
- No additional client software or configuration necessary.
- No need to store your account details (compared to SMTP client 397), since server is fake.
- Can deliver reports automatically. No UI.
- Can fabricate "FROM" field, so it'll match user's e-mail.
- Good for basic support for unsupported web-trackers (see also 153).

Drawbacks:
- No backward feedback - you can't tell customer that this problem is already solved.
- No bug report management.
- Can be blocked by firewall or client's ISP. **This has the highest chances among all e-mail based methods** (some ISPs require to use their SMTP relay servers to send e-mails outside).
- Real e-mail servers may don't like such fake stray e-mail servers.

Typically, you should use either SMTP server or SMTP client 397, but not both methods simultaneously.

Constant: esmSMTPServer.

See also:
- SMTP server method options 319
- SMTP client send method 397
- Other send methods 390
- Selecting send method 55
- Security Considerations 158
- Differences between SSL and TLS modes 588

10.6.6 HTTP upload

HTTP send method uploads bug report via HTTP protocol to web server. It uses HTTP POST method. I.e. it's very similar to filling a form on web page and clicking on "Submit" button. **In order to work, this method requires special script on web server side, which will receive bug reports.**

During HTTP send, EurekaLog sends bug report and additional files (if specified) as standard upload. For example, you can use the following PHP script, which uses standard $_FILES array to process uploaded files:

```
<?php

  foreach($_FILES as $key=>$value)
  {
   $uploaded_file = $_FILES[$key]['tmp_name'];
   // Writable folder:
   $server_dir = 'C:\\upload\\'; // example for Windows
// $server_dir = 'upload/';  // example for *nix
   $server_file = $server_dir.basename($_FILES[$key]['name']);

    // Move the uploaded file to the Server uploaded directory...
    if (move_uploaded_file($uploaded_file, $server_file))
    {
      // Here is your code...
      // You can process uploaded files here, if you want/need to
    }
  }

?>
```

You can place this code to .php file (like upload.php) and place this file to your web server (like http://example.com/upload.php), and then specify this URL in HTTP send options 320 (without "http://").

Warning: be sure to setup adequate maximum upload file limits in your web-server/script configuration. Otherwise sending may fail on large bug reports.

You can also provide additional data via OnCustomWebFieldsRequest event handler. For example, assign such event hanlder:

```
uses
  EEvents;

procedure AddApplicationName(const ACustom: Pointer; ASender: TObject { TELWebSende
begin
  AWebFields.Values['Application'] := AnsiLowerCase(ExtractFileName(ParamStr(0)));
end;

initialization
  RegisterEventCustomWebFieldsRequest(nil, AddApplicationName);
end.
```

Then, you can access your new "Application" field from your script. For PHP it will be $_REQUEST["Application"] or $_POST["Application"].

You don't have to use PHP. You can use script on any other scripting language (like ASP.NET). The above code is just simple example. The point is that such upload is standard file upload, there is no default functionality to handle it, so you need to write some code to handle it. How you do that, what amount of work you'll do - it's up to you.

See also examples below for more advanced script usage.

Advantages:
- Simple setup.
- Highly customizable.

- Little chances to be blocked by firewall (almost everybody have internet access, which means HTTP access).
- Can deliver backward feedback (via custom scripts).
- SSL/TLS support.

Drawbacks:
- No bug report management (by default).
- User e-mail address is optional.
- A lot of custom work to get anything above basic upload functionality.
- Requires hosting or HTTP server.

Currently EurekaLog supports only HTTP Basic-AUTH (basic access authentication) authentication method.

You can test any of your scripts by using HTML page on your web server (say, `test.htm` file):

```
<html>
  <body>
    <form enctype="multipart/form-data" action="upload.php" method="POST">
      <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
      Send this file: <input name="userfile" type="file" />
      <input type="submit" value="Send File" />
    </form>
  </body>
</html>
```

(of course, you must replace "upload.php" with your data; you don't have to place this page on web server, you can run it locally - in this case action parameter will be like this: `action="http://example.com/upload.php"`)

You can also add some debugging output to your scripts. For example (this is modified example of the PHP script above):

```
<?php

  foreach($_FILES as $key=>$value)
  {
    $uploaded_file = $_FILES[$key]['tmp_name'];
    $server_dir = 'c:\\writable\\';
    $server_file = $server_dir . basename($_FILES[$key]['name']);

    echo $uploaded_file;
    echo $server_file;

    if (move_uploaded_file($uploaded_file, $server_file))
    {
      echo 'Ok';
    }
    else
    {
      echo 'Fails';
    }

  }

?>
```

Just don't forget to remove debug code before real using!

It's also possible to get a feedback from upload script 155. Script can set custom message to show and upload status (like "OK", "error" and "bug fixed"). It's done by generating a HTML output with special tags. Currently, EurekaLog supports only 2 tags out-of-the-box: **EurekaLogStatus** and **EurekaLogReply**. To use these tag, you must generate a HTML in your script like this:

```
<?php

  foreach($_FILES as $key=>$value)
  {
    $uploaded_file = $_FILES[$key]['tmp_name'];
    $server_dir = 'c:\\writable\\';
    $server_file = $server_dir . basename($_FILES[$key]['name']);

    move_uploaded_file($uploaded_file, $server_file);

  }

  echo '<html>';
  echo '<head>';
  echo '<META HTTP-EQUIV="CONTENT-TYPE" CONTENT="TEXT/HTML; CHARSET=UTF-8">';
  echo '<title>Bug submission</title>';
  echo '</head>';
  echo '<body>';
  echo 'Thank you!<br />';
  echo '</body>';
  echo '</html>';
?>
```

And you can add these two tags in any place inside this HTML. You can enclose these tags in comments to prevent them from appearing in HTML itself, for example:

```
<?php

  foreach($_FILES as $key=>$value)
  {
    $uploaded_file = $_FILES[$key]['tmp_name'];
    $server_dir = 'c:\\writable\\';
    $server_file = $server_dir . basename($_FILES[$key]['name']);

    move_uploaded_file($uploaded_file, $server_file);

  }
?>
<html>
  <head>
    <META HTTP-EQUIV="CONTENT-TYPE" CONTENT="TEXT/HTML; CHARSET=UTF-8">
    <title>Bug submission</title>
  </head>
<body>
<!--

Place here any EurekaLog-specific tags

<EurekaLogReply>Thank you for your feedback!</EurekaLogReply>

-->
```

```
Thank you!<br />
</body>
</html>
<?php

?>
```

EurekaLogStatus tag must contain an integer value, which represents operation status (`TSendResult` type). Most typically used values are `srSent`, `srBugClosed`, `srInvalidInsert` and `srUnknownError`. `srSent` and `srBugClosed` are considered as success status. All other codes are considered as failure. If this tag isn't used, the `srSent` is the default.

Note: it's important to use numeric value (like, '0', '1', '$1'), not name itself (like 'srSent').

You can use this tag to alter status of sending, but usually you omit this tag.

EurekaLogReply tag contains arbitrary string, which will be used as custom message, describing the operation. If this tag isn't used, the message will be empty. If status of the operation is the success (`srSent` or `srBugClosed`), then this message will appear in `SuccessMessage` field of `TResponse` record. If status is failure - message will appear in `ErrorMessage` field.

Alternatively, you can insert a http:// or https:// link into message. In this case EurekaLog will open a web-browser for this link without showing any other message. Showing HTML page can be used to present "pretty" message, detailed instructions or other advanced messages. For example, if "bug" is not a bug in your software, but problem in run-time configuration, you can insert the URL to your knowledge base article, which describes solution. Another example - you can't solve bug with existing bug report's information. Thus, you close bug and use an URL to web-page, where you ask user to submit more information.

For example:

```
<?php

// place any initialization code here

?>

<html>
  <head>
    <META HTTP-EQUIV="CONTENT-TYPE" CONTENT="TEXT/HTML; CHARSET=UTF-8">
    <title>Bug submission</title>
  </head>
<body>
Thank you!<br />

<!--

<?php

  $fail = TRUE;
  $newbug = TRUE;

  foreach($_FILES as $key=>$value)
  {
    $uploaded_file = $_FILES[$key]['tmp_name'];
    $server_dir = 'c:\\writable\\';
    $server_file = $server_dir . basename($_FILES[$key]['name']);
  echo 'Submitted: ' . basename($_FILES[$key]['name']) . '<br />';

  if (!move_uploaded_file($uploaded_file, $server_file))
   break;
  else
   $fail = FALSE;

 }

 if ($fail)
 {
  echo '<EurekaLogStatus>$8000FFFF</EurekaLogStatus>';
  echo '<EurekaLogReply>Sorry, something goes wrong! Please, report this to admi
 }
 else
 {

  // <- here: determinate if uploaded file(s) contains already fixed bug or not;

  if ($newbug)
  {
   echo '<EurekaLogStatus>0</EurekaLogStatus>';
   echo '<EurekaLogReply>Thank you for your feedback! We will try to fix this a
  }
  else
  {
   echo '<EurekaLogStatus>1</EurekaLogStatus>';
   echo '<EurekaLogReply>Thank you for your feedback! This error was already fi
   echo 'Please, go to www.example.com and download a new version.</EurekaLogRe
  }
 }

?>

-->

</body>
</html>

<?php

// place any finalization code here

?>
```

See also: customizing feedback 155.

Note: remember - **you don't have to use PHP**. PHP is selected only as example language. You can write receiver-script in any language: ASP, .NET, (Win)CGI, etc. Just handle the standard file uploads and return some HTML page. That's all.

For example, you can use FogBugz anonymous report page as HTTP form to receive EurekaLog reports 115.

Constant: wsmHTTP.

See also:

- HTTP method options 320
- FTP upload method 404
- Other send methods 390
- Selecting send method 55
- Security Considerations 158
- FogBugz setup 108
- Customizing feedback 155

10.6.7 FTP upload

FTP send method uploads bug report file to remote folder on FTP server. It's almost like usual file copying, except it "copies" file to remote location.

Advantages:

- Simple and controllable.
- Almost no customizations.
- Minimum efforts to setup (among all web methods, including web-trackers)
- Little chances to be blocked by firewall (almost everybody have internet access, which usually means FTP access too).
- SSL/TLS support (currently not supported by EurekaLog).

Drawbacks:

- May be blocked by firewall.
- No bug report management.
- User e-mail address is optional.
- No backward feedback - you can't tell customer that this problem is already solved.
- Requires hosting or FTP server.

You can test FTP uploads by using any fully-functional FTP client software (web-browsers aren't suitable: they don't support upload functionality).

Notes:

- EurekaLog doesn't support SFTP protocol; only FTP is supported.
- Currently EurekaLog supports only FTP; FTPS is not supported.

Constant: wsmFTP.

See also:

- FTP method options 321
- HTTP upload method 398
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

10.6.8 FogBugz

This send method creates a new issue on FogBugz bug tracker (see demo).

A bug tracking system is a software application that is designed to help quality assurance and programmers keep track of reported software bugs in their work. It may be regarded as a type of issue tracking system.

Many bug-tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly. Other systems are used only internally in a company or organization doing software development. Typically bug tracking systems are integrated with other software project management applications.

Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products. Consistent use of a bug or issue tracking system is considered one of the "hallmarks of a good software team".

This method requires some preparation actions, before you can use it:

1. You need to get web server with ASP.NET and database running or have a hosting with ASP.NET and database support. You can get it with FogBugz, if you buy FogBugz as service.
2. (Optional) You need to download FogBugz and upload it on web-server (either hosting or self-server). This step is optional, if you use FogBugz hosting services.
3. (Optional) You need to run through FogBugz installer. This step is optional, if you use FogBugz hosting services.

Important note: EurekaLog uses FogBugz XML API to access FogBugz. BugzScout is not used by EurekaLog with "FogBugz" send method, however you can use BugzScout with HTTP send method 115.

4. You need to setup FogBugz:
   - create projects (for each of your products).
   - create "reporter" account (warning: don't use administrator account for reporting issues).
5. Fill FogBugz details in your EurekaLog-enabled application.

You also need to setup sending options and test sending 108. To test send - place a button, which raises exception. Let EurekaLog handle it and perform sending. Ensure that it's working as expected. Send report again to check how duplicates are handled. Adjust any options, if needed. Change status of issue to "closed" and retry sending. Ensure that it works as expected.

Advantages:

- Powerfull and customizable.
- Bug report management.
- Can be get as paid service with 0 setup time.
- SSL/TLS support.
- **Easy to use.**

Drawbacks:

- Paid software.
- May be blocked by firewall.
- Requires setup 108.
- Requires hosting OR web (http) server + database service.
- User e-mail address is optional.
- Requires setup for each your project.
- Setup options and access rights may be not detailed enough.

Note: you may consider using HTTP upload method 398 for FogBugz instead of using FogBugz API. See FogBugz setup 108 for more detailed description; see FogBugz: using HTTP upload 115 for detailed manual on HTTP Upload setup for FogBugz.

Constant: wsmFogBugz.

See also:

- FogBugz:
  - Feature list
  - Demo
  - Installation instructions
  - Hosting
  - Self-server version
  - Comparison of issue-tracking systems
- FogBugz method options 322
- Managing bug reports 105
- FogBugz setup 108
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

- Using HTTP upload with FogBugz 115

10.6.9 Mantis

This send method creates a new issue on Mantis bug tracker (see demo).

A bug tracking system is a software application that is designed to help quality assurance and programmers keep track of reported software bugs in their work. It may be regarded as a type of issue tracking system.

Many bug-tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly. Other systems are used only internally in a company or organization doing software development. Typically bug tracking systems are integrated with other software project management applications.

Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products. Consistent use of a bug or issue tracking system is considered one of the "hallmarks of a good software team".

This method requires some preparation actions, before you can use it:

1. You need to get web server with PHP and database running or have a hosting with PHP and database support.
2. (Optional) You need to download Mantis and upload it on web-server (either hosting or self-server). This step is optional, if you use Mantis hosting services.
3. (Optional) You need to run through Mantis installer. This step is optional, if you use Mantis hosting services.

Important note: EurekaLog uses SOAP API to access Mantis. SOAP API is enabled by default, but you need to make sure it was not disabled. Please refer to Mantis documentation for more information. Also be aware that SOAP implementation in Mantis has several bugs - see our knowledgebase.

4. You need to setup Mantis:
   - create projects (for each of your products).
   - create "reporter" account (warning: don't use administrator account for reporting issues).
   - create custom field for "counting" bug reports.
5. Fill Mantis details in your EurekaLog-enabled application.

You also need to setup sending options and test sending 119. To test send - place a button, which raises exception. Let EurekaLog handle it and perform sending. Ensure that it's working as expected. Send report again to check how duplicates are handled. Adjust any options, if needed. Change status of issue to "closed" (or "resolved") and retry sending. Ensure that it works as expected.

Advantages:

- **Powerfull and customizable.**
- Bug report management.
- Freeware (hosting cost not included).
- SSL/TLS support.
- Full control over access rights and user/project options.

Drawbacks:

- May be blocked by firewall.
- Requires setup 119.
- Requires hosting OR web (http) server + database service.
- User e-mail address is optional.
- Requires setup for each your project.
- Huge amount of options is hard to configure.

Constant: wsmMantis.

See also:

- Mantis:
  - Feature list
  - Demo
  - Evaluation (Instant Mantis)
  - Hosting services
  - Installation instructions
  - FAQ
  - Comparison of issue-tracking systems
- Mantis method options 327
- Managing bug reports 105
- Mantis setup 119
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

10.6.10 BugZilla

This send method creates a new issue on BugZilla bug tracker (see demo).

A bug tracking system is a software application that is designed to help quality assurance and programmers keep track of reported software bugs in their work. It may be regarded as a type of issue tracking system.

Many bug-tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly. Other systems are used only internally in a company or organization doing software development. Typically bug tracking systems are integrated with other software project management applications.

Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products. Consistent use of a bug or issue tracking system is considered one of the "hallmarks of a good software team".

This method requires some preparation actions, before you can use it:

1. You need to get web server with Perl and database running or have a hosting with Perl and database support.
2. You need to download BugZilla and upload it on web-server (either hosting or self-server).
3. You need to run through BugZilla installer.

Important note: EurekaLog uses XML-RPC interface to access BugZilla. Make sure that you have installed all required Perl packages (usual Perl package dependencies are: SOAP-Lite, XMLRPC-Lite, and Test-Taint) and enabled XML-RPC. Please refer to BugZilla documentation for more information.

4. You need to setup BugZilla:
   - create projects (for each of your products).
   - create "reporter" account (warning: don't use administrator account for reporting issues).
- create custom field for "counting" bug reports.

5. Fill BugZilla details in your EurekaLog-enabled application.

You also need to setup sending options and test sending 134. To test send place a button, which raises exception. Let EurekaLog handle it and perform sending. Ensure that it's working as expected. Send report again to check how duplicates are handled. Adjust any options, if needed. Change status of issue to "closed" and retry sending. Ensure that it works as expected.

Advantages:

- Powerfull and customizable.
- Bug report management.
- Freeware (hosting cost not included).
- SSL/TLS support.

Drawbacks:
- May be blocked by firewall.
- Requires setup 134.
- Requires hosting OR web (http) server + database service.
- User e-mail address is optional.
- Requires setup for each your project.
- Hard to install.
- **API support should be installed separately.**

Constant: wsmBugZilla.

See also:
- BugZilla:
  - Feature list
  - Demo
  - Installation instructions
  - FAQ
  - Comparison of issue-tracking systems
- BugZilla method options 331
- Managing bug reports 105
- BugZilla setup 134
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

10.6.11 JIRA

This send method creates a new issue on JIRA bug tracker (see demo).

A bug tracking system is a software application that is designed to help quality assurance and programmers keep track of reported software bugs in their work. It may be regarded as a type of issue tracking system.

Many bug-tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly. Other systems are used only internally in a company or organization doing software development. Typically bug tracking systems are integrated with other software project management applications.

Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products. Consistent use of a bug or issue tracking system is considered one of the "hallmarks of a good software team".

This method requires some preparation actions, before you can use it:
1. You need to get server with JAVA installed (Windows/Linux/Solaris only) and database running or have a hosting with similar requirements.
2. You need to download JIRA and upload it on server (either hosting or self-server).
3. You need to run through JIRA installer.

Important note: EurekaLog uses JSON API to access JIRA. Make sure you have enabled remote access via API in JIRA options. Please refer to JIRA documentation for more information.

4. You need to setup JIRA:
   - create projects (for each of your products).
   - create "reporter" account (warning: don't use administrator account for reporting issues).
- create custom field for "counting" bug reports.
5. Fill JIRA details in your EurekaLog-enabled application.

You also need to setup sending options and test sending 143. To test send place a button, which raises exception. Let EurekaLog handle it and perform sending. Ensure that it's working as expected. Send report again to check how duplicates are handled. Adjust any options, if needed. Change status of issue to "closed" and retry sending. Ensure that it works as expected.

Advantages:
- Powerfull and customizable.
- Bug report management.
- Can be get as paid service with 0 setup time.
- SSL/TLS support.
- Setup options are easy to use.

Drawbacks:
- Paid software.
- May be blocked by firewall.
- Requires setup 143.
- Requires hosting OR web (http) server + database service.
- User e-mail address is optional.
- Requires setup for each your project.
- Setup options and access rights may be not detailed enough.

Constant: wsmJIRA.

See also:
- JIRA:
  - Feature list
  - Demo
  - Installation instructions
  - FAQ
  - Comparison of issue-tracking systems
- JIRA method options 335
- Managing bug reports 105
- JIRA setup 143
- Other send methods 390
- Selecting send method 55
- Security Considerations 158

10.7 Debug information providers

Debug information provider is a service class that can extract information about code from the particular format of debug information in the executable module or external file. EurekaLog uses debug information providers to extract information required to build human-readable call stacks.

Available providers are:

Provider	Type	Description
EurekaLog 410	build-in	Standard provider for EurekaLog's own debug information format. Injected.
MAP files 410	plugin	Provider for .map files format. External .map files only. Can not be injected. Delphi or C++ Builder .map files only. 3rd party .map files are not supported.
Turbo Debugger 411	plugin	Provider for Turbo Debugger format (TD32). Injected or .tds files.
DLL exports 411	plugin	Provider for DLL with export table. Heuristic.
JCL 412	plugin	Provider for JEDI (JclDebug) format. Injected or .jbdg files.
Microsoft 412	plugin	Provider for Microsoft formats. Requires external DLL. External .dbg/.pdb files only. Can not be injected.
madExcept 413	plugin	Experimental provider for madExcept debug format. Doesn't requires madExcept installed.

See also:

- Debug information settings 243
- Using debug information converters 516
- Configuring call stack 48

10.7.1 EurekaLog

EurekaLog debug information provider allows your application to extract information from EurekaLog debug information format. This provider also supplies stored module options for EurekaLog code. This provider is build-in and always enabled, you don't need special actions to enable it.

EurekaLog uses ZLib compression and TEA encryption to store debug information. Debug information is never stored in clear text - even if no password is set. Only unit, class, routine names and line numbers are stored. Debug information is injected into executable file. Format itself is a custom binary format.

Overall size is compact. Exact estimated depends on your detalization level. Format is binary, so it's fast.

This is default, recommended and best-option for EurekaLog-enabled applications.

Note: EurekaLog information is generated from .map and (optionally) .tds files. Sometimes other sources 496 can be used for 3rd party compilers. These service files (i.e. .map/.tds) must be generated, but they are no longer needed after compilation. They will be deleted unless you explicitly ask to keep them 234. You don't need to distribute any additional files when you use default EurekaLog debug information format.

See also:

- Debug information providers options 355
- Configuring call stack 48
- Project settings for debugging 225
- List of debug information providers 409
- EurekaLog for shareware developers 585

10.7.2 .map file

MAP file provider allows your application to extract information from .map files.

.map files are text human-readable files which are generated by IDE linker during project compilation and linking process. .map files contains debug information about compiled module in clear text form. .map files contains segments information, units, class, routine names and line numbers. Level of detalization for .map files is set in project options.

Use .map files when you want to use other debug tools which can understand .map files, but do not understand EurekaLog debug information format. Consider using EurekaLog debug format instead of .map files. .map files are large (often they are larger than executable itself). They are text files, so parsing is required - which is slow. There is no protection, no packing, no encryption.

You'll need to distribute .map files with your application.

Notes:

- It's highly recommended to convert .map files to EurekaLog debug format. You are able to do such conversion without DLL recompilation as explained in this article 426.
- .map files do not have a strict format. .map files are defined as "human-readable plain text files in free form that indicate the relative offsets of functions for a given version of a compiled binary". EurekaLog is able to parse .map files produced by Delphi and C++ Builder linkers. EurekaLog is not able to parse .map files produced by other compilers/linkers/tools.

See also:

- Debug information providers options 355

- Configuring call stack 48
- Project settings for debugging 225
- List of debug information providers 409
- Using debug information converters 516
- Post-processing without (re)compilation 426

10.7.3 TD32

Turbo Debugger provider allows your application to extract information from TD32 debug information source. It can be external .tds file or injected information.

TD32 debug information is used by many debuggers. It's not required by Delphi/Builder IDE, but it's often used for interoperability reasons. It's de-facto standard for Pascal-based debuggers. TD32 contains not only names and line numbers, but also extensive information about variables, arguments, calling conventions and so on. However, EurekaLog uses only location information.

Use TD32 information when you want to use other debug tools which can understand TD32 format, but do not understand EurekaLog debug information format. Consider using EurekaLog debug format instead of TD32 format. TD32 format is extremely large (sometimes a x10 of original executable). They are binary files. There is no protection, no packing, no encryption. All debug information is stored in clear text.

You'll need to distribute .tds files with your application or you can inject TD32 information into executable (controlled by project options).

Note: It's highly recommended to convert .tds files to EurekaLog debug format. You are able to do such conversion without DLL recompilation as explained in this article 426.

See also:

- Debug information providers options 355
- Configuring call stack 48
- Project settings for debugging 225
- List of debug information providers 409
- Using debug information converters 516

10.7.4 Exports table

DLL exports provider allows your application to extract heuristic information about functions in DLL based on export table.

This provider is heuristic. It doesn't give you exact information about functions. It can detect start address of the function, but not its size. So it uses guessing about sizes.

Note: DLL Exports provider may show entries like "(possible fnMSSample+132)". Such text means that there are some JMP or RET instructions between start of the function and actual address in a call stack. This means:

- [Positive] Address belongs to the specified function. JMP/RET instruction may be part of the function's logic (such as try/except block);
- [False-positive] Address does not belong to the specified function. JMP/RET instruction marks the end of the function. Address itself lies within some other internal/unknown function after the specified function.

Number after "+" sign indicate byte offset between function's start and call stack address.

No additional information is needed to work, therefore this provider can work on **any** DLL.

This provider is good to have in any application as last resort provider - for cases when executable modules (DLL/BPL) do not have any other debug information source. It's highly recommended to create debug information source for DLLs when possible.

The common alternatives for this provider:

- DLL can be post-processed by EurekaLog 410 with "DLL" profile 368;
- DLL can be post-processed by JCL 412;
- DLL can be post-processed by madExcept 413;

- DLL can supply .map 410/.tds 411 files (this is only useful for IDEs without any exception tracer tool installed);
- DLL can supply PDB/DBG files 412;
- Non-Embarcadero DLL can be post-processed by EurekaLog based on output from 3rd party compiler 496;

Note: you can ask Microsoft for debug information about system DLLs. See this article 504 for more information.

See also:

- Debug information providers options 355
- Configuring call stack 48
- List of debug information providers 409
- Using Microsoft DbgHelp DLL 504
- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
- Using EurekaLog with non-Embarcadero compilers (Microsoft Visual Studio, etc.) 496

10.7.5 JCL

JCL provider allows your application to extract debug information from JEDI debug information format. It can be external .jdbg file or injected into executable.

JEDI Code Library (JCL) is a popular freeware open-source code library for Delphi, Kylix, C++ Builder and Free Pascal. It contains JclDebug and JclHookExcept units which can be used to build your own exception tracer from scratch.

Use JCL format when you want to use other Delphi tools which can understand JCL format, but do not understand EurekaLog debug information format. JCL debug information format is de-facto standard of debug information in run-time. Many Delphi-specific tools are able to work with it. It's a custom binary format, which contains unit, class, routines names, and line numbers. It uses ZLib packing to minimize final size. There is no password protection. Overall this format is very similar to EurekaLog debug format.

You'll need to distribute .jdbg files with your application or inject debug information into executables (controlled by JCL project options).

Note: many versions of Delphi and C++ Builder ships RTL and VCL packages with JCL debug information. You can find it in .jdbg files near .bpl files in \Bin folder. So, if you build a packaged application - you probably should enable JCL debug information provider and distribute RTL and VCL .bpl/.jdbg files along with your application. If you want the same for non-Delphi system DLLs - use Microsoft debug provider 412.

See also:

- Debug information providers options 355
- Using EurekaLog with DLLs post-processed by 3rd party tools (such as JCL, madExcept, etc.) 495
- Configuring call stack 48
- Microsoft debug provider 412
- List of debug information providers 409

10.7.6 Dbg/Pdb

Microsoft debug information provider allows your application to extract debug information from Microsoft's formats. It can be DBG or PDB format. It's stored in standalone .dbg and .pdb files.

Use Microsoft format when you want to use other non-Delphi related tools which can understand MS format, but do not understand Delphi debug formats (such as Process Explorer or WinDbg). You can create debug information in MS format for your project by using freeware map2dbg and tds2pdf tools.

This provider requires Microsoft DbgHelp.dll to be present. You can get it either from Microsoft Debugging Tools or from \Bin and \Bin64 subfolders of EurekaLog installation. You should deploy this DLL with your application. Use DLL of corresponding bitness. Do not install this DLL to system folders.

Note: you can use MS debug format to get information for system DLLs. By default Windows comes in release version without debug information for its DLLs (so-called "free build"). This prevents you from getting proper information. In fact, you only can get heuristic information based on DLL exports 411. However, you can ask Microsoft for debug information and get full coverage for Microsoft DLLs. See this article 504 for more information.

An alternative for this provider and your own custom DLLs is converting PDB/DBG debug information into EurekaLog debug information as explained in this article 496.

See also:
- Debug information providers options 355
- Configuring call stack 48
- Using Microsoft DbgHelp DLL 504
- Using EurekaLog with DLLs compiled by 3rd party compilers (such as Microsoft Visual Studio, etc.) 496
- JCL debug provider 412
- DLL exports provider 411
- List of debug information providers 409
- Using debug information converters 516

10.7.7 madExcept

madExcept provider allows your application to extract debug information from madExcept debug information format. It can be external .mad file or injected into executable.

madExcept is an exception tracer tool. Use madExcept format when you want to use your existing executables compiled with madExcept in EurekaLog-enabled application.

You'll need to distribute .mad files with your application or inject debug information into executables (controlled by madExcept project options).

Note: this provider is experimental. It may not work with all version of madExcept formats.

See also:
- Debug information providers options 355
- Using EurekaLog with DLLs post-processed by 3rd party tools (such as JCL, madExcept, etc.) 495
- Configuring call stack 48
- List of debug information providers 409

10.8 Variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They can be said in some sense to create the operating environment in which a process runs. For example, an environment variable with a standard name can store the location that a particular computer system uses to store temporary files - this may vary from one computer system to another. A process which invokes the environment variable by (standard) name can be sure that it is storing temporary information in a directory that exists and is expected to have sufficient space.

You can use environment variables in any text values in your project settings 225. You can insert variable by using "Variables" window 228. Variable is inserted as special tag. When you run your application at run-time, any variable value will be replaced with actual value, which is calculated at run-time.

For example, if you set your folder for saving bug-report to "%APPDATA%\MyBugReports" then your bug reports may be saved to (few examples):
- C:\Users\UserName\AppData\Roaming\MyBugReports\
- C:\Documents and Settings*UserName*\Application Data\MyBugReports\

- `C:\WINDOWS\system32\config\systemprofile\AppData\Roaming\MyBugReports\`
- `C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\MyBugReports\`

Depending on real value of %APPDATA% variable at run-time.

Note: variables names are case-insensitive.

EurekaLog

EurekaLog expands system variables set by the following special pseudo-variables. They can be used in the same way as any other usual environment variable.

Warning: don't forget that you need to enclose variable name into '%', for example for `_BugID` variable you must use '*%_BugID%*' text (without quotes).

EurekaLog's variables:

- **_ExceptType** - exception's class. For example: 'EAccessViolation'. It will be replaced with empty string if there is no exception in question.
- **_ExceptMsg** - exception's message. For example: 'Access violation at address 00000000. Read at address 000000'. It will be replaced with empty string if there is no exception in question.
- **_CallStack** - compact call stack of exception (multi-line). It will be replaced with empty string if there is no exception in question.
- **_BugID** - exception's BugID 421. For example: '47A10000'. It will be replaced with empty string if there is no exception in question.
- **_Reproduce** - "steps to reproduce" text as it was entered by user (in error dialog). This is arbitrary multi-line text. Empty by default.
- **_BugReport** - full bug report text (multi-line). It will be replaced with empty string if there is no exception in question.
- **_LineBreak** - the simple line break (#13#10 or #10 - depending on your platform). It's useful to enter multi-line texts in single-line edit control.
- **_*XYZ*ModulePath** - path to executable file without trailing path delimiter. For example: 'C:\Program Files\My Product'. *XYZ* part selects module in question.
- **_*XYZ*ModuleName** - executable file name. For example: 'Project1.exe'. *XYZ* part selects module in question.
- **_*XYZ*ModuleDesc** - description of executable file. It's extracted from file's version information. *XYZ* part selects module in question.
- **_*XYZ*ModuleVer** - version of executable file. For example: '1.0.2.0'. It's extracted from file's version information. *XYZ* part selects module in question.

Where ***XYZ*** can be:

- **Main** - .exe file of running process.
- **Except** - module which has raised exception (applicable only if there is exception in question).
- **This** - your project (.exe or .DLL).

If your project is .exe - "This" will be the same as "Main". If exception was raised in your project (and not in some other DLL) - "Except" will be the same as "This". If your application consists only of single .exe and never catches exceptions from other DLLs - then all 3 values will be the same. We suggest to use "Except" case as default (e.g. like `%_ExceptModuleName%`).

The following variables are available only for IDE and compilation options (like pre/post build events, external configuration path, localization path, etc.):

- **_IDEProject** - file name of project file. Like `C:\Projects\MyProject.dpr` for Delphi 7 or `C:\Projects\MyProject.dproj` for Delphi XE.
- **_IDESource** - file name of project source file. Like `C:\Projects\MyProject.dpr` for both Delphi 7 and Delphi XE.
- **_IDEConfig** - file name of options file. Like `C:\Projects\MyProject.dof` for Delphi 7, `C:\Projects\MyProject.dproj` for Delphi XE or `C:\Projects\Options\External.eof` for external configuration 443.
- **_IDETarget** - final compiled executable file. Like `C:\Projects\MyProject.exe` or `C:\Projects\MyProject.dll`.

- **_IDESrc** - folder with trailing path delimiter, containing project's files. Like `C:\Projects\`
- **_IDEDst** - folder with trailing path delimiter, containing compiled executable. Like `C:\Projects\` for Delphi 7 or `C:\Projects\Win32\Debug\` for Delphi XE.

System/Windows variables:

- **CSIDL_*ABC*** - special folder identified by CSIDL *ABC*.
- **FOLDERID_*DEF*** - special folder identified by FOLDERID *DEF*.

Where ***ABC*** can be:

- Name of CSIDL value. For example: APPDATA or COMMON_APPDATA or LOCAL_APPDATA, so full variable name will be `%CSIDL_COMMON_APPDATA%`, `%CSIDL_APPDATA%` or `%CSIDL_LOCAL_APPDATA%`. See this topic in MSDN to get the full list of all possible CSIDL values.
- Numeric value of CSIDL. For example: 26 or $001a, so full variable name will be `%CSIDL_26%` or `%CSIDL_$1a%`.

See also: %APPDATA% and other Windows environment variables below.

Where ***DEF*** can be:

- Name of FOLDERID value. For example: RoamingAppData, ProgramData, LocalAppData or LocalAppDataLow, so full variable name will be `%FOLDERID_RoamingAppData%`, `%FOLDERID_ProgramData%`, `%FOLDERID_LocalAppData%` or `%FOLDERID_LocalAppDataLow%`. See this topic in MSDN to get the full list of all possible FOLDERID values.
- GUID-value of FOLDERID. For example: {3EB685DB-65F9-4CF6-A03A-E3EF65729F3D}, so full variable name will be `%FOLDERID_{3EB685DB-65F9-4CF6-A03A-E3EF65729F3D}%`.

See also: %APPDATA% and other Windows environment variables below.

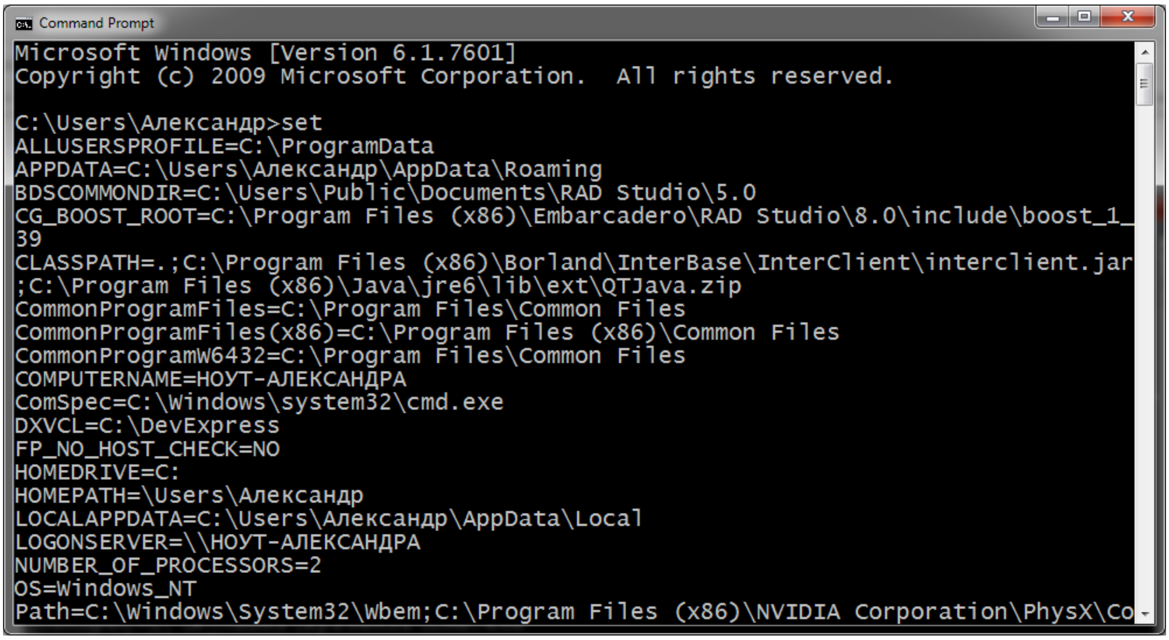
Note: FOLDERID values are available only for Windows Vista+. CSIDL values are available on all Windows platforms.

Warning: if variable doesn't exist - it will be replaced with empty string. Be careful to use variables which are available only on certain platform or environment.

You can also use any system environment variable apart from EurekaLog-specific variables above. Just use %*system-variable-name*% - where *system-variable-name* is a name of system variable (see below). You can use any system variable - even if it's not listed in "Variables" window 228. "Variables" window lists only build-in pseudo-variables and some commonly used system variables.

Windows

In Windows, the set command without any arguments displays all environment variables along with their values:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Александр>set
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Александр\AppData\Roaming
BDSCOMMONDIR=C:\Users\Public\Documents\RAD Studio\5.0
CG_BOOST_ROOT=C:\Program Files (x86)\Embarcadero\RAD Studio\8.0\include\boost_1_
39
CLASSPATH=.;C:\Program Files (x86)\Borland\InterBase\InterClient\interclient.jar
;C:\Program Files (x86)\Java\jre6\lib\ext\QTJava.zip
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=НОУТ-АЛЕКСАНДРА
ComSpec=C:\Windows\system32\cmd.exe
DXVCL=C:\DevExpress
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Users\Александр
LOCALAPPDATA=C:\Users\Александр\AppData\Local
LOGONSERVER=\\НОУТ-АЛЕКСАНДРА
NUMBER_OF_PROCESSORS=2
OS=Windows_NT
Path=C:\Windows\System32\Wbem;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Co
```

Example of environment variables list

To set a variable to a particular value, use:

```
set VARIABLE=value
```

However, this change is temporal, as "set" command makes changes to current environment variables, changes are not saved to system settings - thus such changes will be lost after restart.

Permanent changes to the environment variable can be achieved through GUI (Control Panel/System/Advanced/Environment Variables), using the setx.exe application, and editing the registry - HKCU\Environment (for user specific variables) or HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment (for system variables) - this is not recommended for novices. The setx.exe application is part of Windows since Windows Vista. It is also a part of Windows Resource Kit for previous Windows versions.

Use:

```
setx VARIABLE value
```

to change permanently user's (local) environment variables. Use:

```
setx VARIABLE value /m
```

to change permanently system's (global) environment variables. The later requires administrative access.

Please note that environment variables are stored per-process and are inherited from parent processes. Changing environment variables in system configuration will not change environment variables of already running processes. For example, using "set" command in console will change environment block of console process - because "set" is internal command. On the other hand, using "setx" will not alter environment block of console process, because setx.exe is an external application and have no access to environment variables of parent process. In most cases processes has to be restarted to get notified of changes in stored environment variables. However, some processes may react to broadcasts of WM_SETTINGCHANGE message, and reload/update environment variable block.

Use SetEnvironmentVariable function to alter environment variable(s) in your own process.

To see the current value of a particular variable, use:

```
echo %VARIABLE%
```

or

```
set VARIABLE
```

Use GetEnvironmentVariable function to retrieve value of environment variable in your own process.

To delete a variable, the following command is used:

```
set VARIABLE=
```

or

```
setx VARIABLE ""
```

```
setx VARIABLE "" /m
```

(depending on local/global scope)

Unfortunately, the setx.exe application will only clear variable, but not delete it completely. However, the final result is the same (e.g. empty value).

Note: The names of environment variables are case-insensitive in Windows.

Important Note: Windows environment variables are specific to user account. Each user account has its own set of variables. EurekaLog will use variables of user running your executable. For example, if you are writing a Win32 Service application - such application is run by "Local System" account by default. "Local System" account has its own environment variables values, which are different from environment variables values of your current interactive user account.

Standard Windows environment variables

1. General pseudo-variables - these variables generally expand to discrete values, such as the current working directory, the current date, or a random number. They look like environment variables, but they are not:
   - **%CD%** - this variable points to the current directory. Equivalent to the output of the command cd when called without arguments.
   - **%DATE%** - this variable expands to the current date. The date is displayed according to the current user's date format preferences. EurekaLog also offers **%DATEFMT%** pseudo-variable, which represents current date in fixed format.
   - **%ERRORLEVEL%** - this variable points to the current error level. If there was an error in the previous command, this is what you need to check against to find out about that. Not applicable outside of cmd shell.
   - **%RANDOM%** - this variable returns a random number between 0 and 32767.
   - **%TIME%** - this variable points to the current time. The time is displayed according to the current user's time format preferences. EurekaLog also offers **%TIMEFMT%** pseudo-variable, which represents current time in fixed format.

2. System path variables - these variables refer to locations of critical operating system resources, and as such generally are not user-dependent:
   - **%AllUsersProfile%** (%PROGRAMDATA% for Windows Vista, Windows 7) - expands to the full path to the `All Users` profile directory. This profile contains resources and settings that are used by all system accounts. Shortcut links copied to the All Users' Start menu or Desktop folders will appear in every user's Start menu or Desktop, respectively.
   - **%ComSpec%** this variable contains the full path to the command processor, `cmd.exe`.
   - **%PATH%** - this variable contains a semicolon-delimited (do not put spaces in between) list of directories in which the command interpreter will search for an executable file that matches the given command.
   - **%ProgramFiles%** - this variable points to Program Files directory, which stores all

the installed applications. The default is C:\Program Files. In 64-bit editions of Windows (XP, 2003, Vista), there are also `%ProgramFiles(x86)%` which defaults to C:\Program Files (x86) and `%ProgramW6432%` which defaults to C:\Program Files. The `%ProgramFiles%` itself depends on whether the process requesting the environment variable is itself 32-bit or 64-bit (this is caused by Windows-on-Windows 64-bit redirection).
- **%CommonProgramFiles%** - this variable points to Common Files directory.
- **%SystemDrive%** - is the drive upon which the system folder was placed. Also see next item.
- **%SystemRoot%** - is the location of the system folder, including the drive and path. The drive is the same as `%SystemDrive%` and the default path on a clean installation depends upon the version of the operating system. By default, on a clean installation: Windows NT 5.1 (Windows XP) and newer versions use `\WINDOWS`. Windows NT 5.0 (Windows 2000) use \WINNT.
- **%TEMP%** and **%TMP%** - these variables contain the path to the directory where temporary files should be stored.
- **%WinDir%** - this variable points to the Windows directory. It is identical to the `%SystemRoot%` variable above on modern systems.

3. User management variables - these variables store information related to resources and settings owned by various user profiles within the system. As a general rule, these variables do not refer to critical system resources or locations that are necessary for the OS to run:
   - **%AppData%** - contains the full path to the Application Data folder of the logged-in user.
   - **%LOCALAPPDATA%** - this variable is the temporary files of Applications. Its uses include storing of Desktop Themes, Windows Error Reporting, Caching and profiles of web browsers.
   - **%UserDomain%** - holds the name of the Workgroup or Windows Domain to which the current user belongs. The related variable, %LOGONSERVER%, holds the hostname of the server that authenticated the current user's logon credentials (name and password). For Home PCs, and PCs in a Workgroup, the authenticating server is usually the PC itself. For PCs in a Windows Domain, the authenticating server is a domain controller (a primary domain controller, or PDC, in Windows NT 4-based domains).
   - **%UserName%** - is the name/login of current user.
   - **%UserProfile%** - is the location of the current user's profile directory, in which is found that user's HKCU registry hive (NTUSER). Users can also use the %USERNAME% variable to determine the active users login identification.

Notes:
- Some variables exist only on modern versions of Windows and aren't available on old/previous versions.
- Some variables exist only in 64-bit edition of Windows. For example, `%ProgramW6432%` are not supported in 32-bit operating systems.
- Some of the variables depend on the bitness of the caller and thus lead to different results, depending on whether the caller is 32- or 64-bit application. For example, program installers typically use `%ProgramFiles%` to install the application in the Program Files folder. Thus, the 64-bit installer will install the program in `C:\Program Files\`, and 32-bit installer - in `C:\Program Files (x86)\`.
- Some variables are valid only for 32-bit or 64-bit processes. Refer to MSDN for more information.
- CSIDL and FOLDERID values are special EurekaLog's pseudo-variables. They always use WinAPI to obtain folder's path. Variables like %APPDATA% are usual environment variables, so they can be altered by parent process.

Important Note: Windows environment variables are specific to user account. Each user account has its own set of variables. EurekaLog will use variables of user running your executable. For example, if you are writing a Win32 Service application - such application is run by "Local System" account by default. "Local System" account has its own environment variables values, which are different from environment variables values of your current interactive user account.

Examples of default values for Windows environment variables

Variable	Windows XP and below	Windows Vista and above
%ALLUSERSPROFILE%	C:\Documents and Settings\All Users	C:\ProgramData
%APPDATA%	C:\Documents and Settings*{username}*\Application Data *(normal user account)* C:\WINDOWS\system32\config\systemprofile\Application Data\ *(local system account)*	C:\Users\*{username}*\AppData\Roaming *(normal user account)* C:\WINDOWS\system32\config\systemprofile\AppData *(local system account)*
%COMPUTERNAME%	*{computername}*	*{computername}*
%COMMONPROGRAMFILES%	C:\Program Files\Common Files	C:\Program Files\Common Files
%COMMONPROGRAMFILES(x86)%	C:\Program Files (x86)\Common Files	C:\Program Files (x86)\Common Files
%COMSPEC%	C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe
%HOMEDRIVE%	C:	C:
%HOMEPATH%	\Documents and Settings*{username}*	\Users*{username}*
%LOCALAPPDATA%		C:\Users\*{username}*\AppData\Local
%LOGONSERVER%	\\*{domain_logon_server}*	\\*{domain_logon_server}*
%PROGRAMFILES%	C:\Program Files	C:\Program Files
%PROGRAMFILES(X86)%	C:\Program Files (x86)	C:\Program Files (x86)
%SystemDrive%	C:	C:
%SystemRoot%	C:\Windows	C:\Windows
%TEMP% and %TMP%	C:\Documents and Settings*{username}*\Local Settings\Temp *(normal user account)* C:\Windows\Temp *(local system account)*	C:\Users\*{username}*\AppData\Local\Temp *(normal user account)* C:\Windows\Temp *(local system account)*
%USERDOMAIN%	*{userdomain}*	*{userdomain}*
%USERNAME%	*{username}*	*{username}*
%USERPROFILE%	C:\Documents and Settings*{username}*	C:\Users\*{username}*
%WINDIR%	C:\Windows	C:\Windows
%PUBLIC%		C:\Users\Public
%PROGRAMDATA%		C:\ProgramData
%PSModulePath%		C:\system32\WindowsPowerShell\v1.0\Modules\

See also:

- Using variables 228

Part XI

11 Advanced topics

Advanced articles covers different topics to get you better understanding of EurekaLog and exception tracers in general. These articles are not required for basic knowledge, so new users may skip these articles to read them later.

11.1 BugID

BugID is a 4-byte unsigned integer (LongWord) which is used to identify exception, for example: 76BA0000. Two exception with the same BugID value are considered to be the same. Two bug reports are considered to be duplicates of each other if they are bug reports about exceptions with the same BugID. BugID is used to establish uniqueness for many EurekaLog features, for example:

- "Do not save duplicate exceptions" option 264 identifies unique reports via BugID;
- Sending 302 identifies unique reports via BugID;
- **EurekaLog Viewer** identifies unique reports via BugID.

BugID consists of two parts. Hi-word is defined by EurekaLog. Low-word is not used by EurekaLog and it's left for your customizations. For example, if BugID is 76BA0000, then hi-word (defined by EurekaLog) is $76BA, low-word (defined by you) is $0000.

Hi-word of BugID is a CRC16 value of string ("BugID ident") which includes:

- application and module identification
- exception class (including additional error number for Win32 and OLE errors)
- 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.

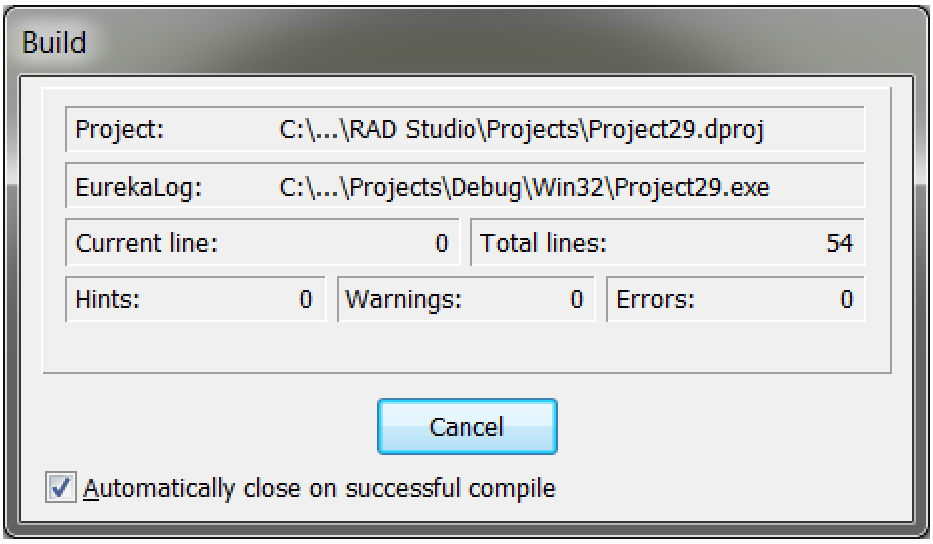
Note:

See also

- ExceptionInfo.BugID property
- Exception filters 344
- Customization example

11.2 Compiling your project with EurekaLog

EurekaLog requires post-processing of your file in order to work 38. If you compile your project inside IDE, then this post-processing is performed by IDE expert:



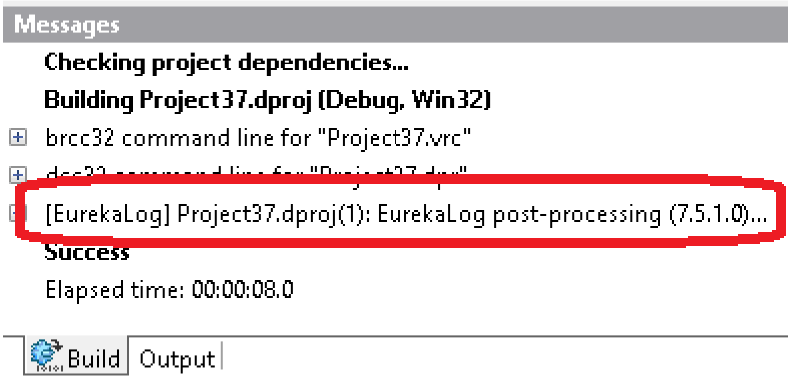
Build

Project: C:\...\RAD Studio\Projects\Project29.dproj

EurekaLog: C:\...\Projects\Debug\Win32\Project29.exe

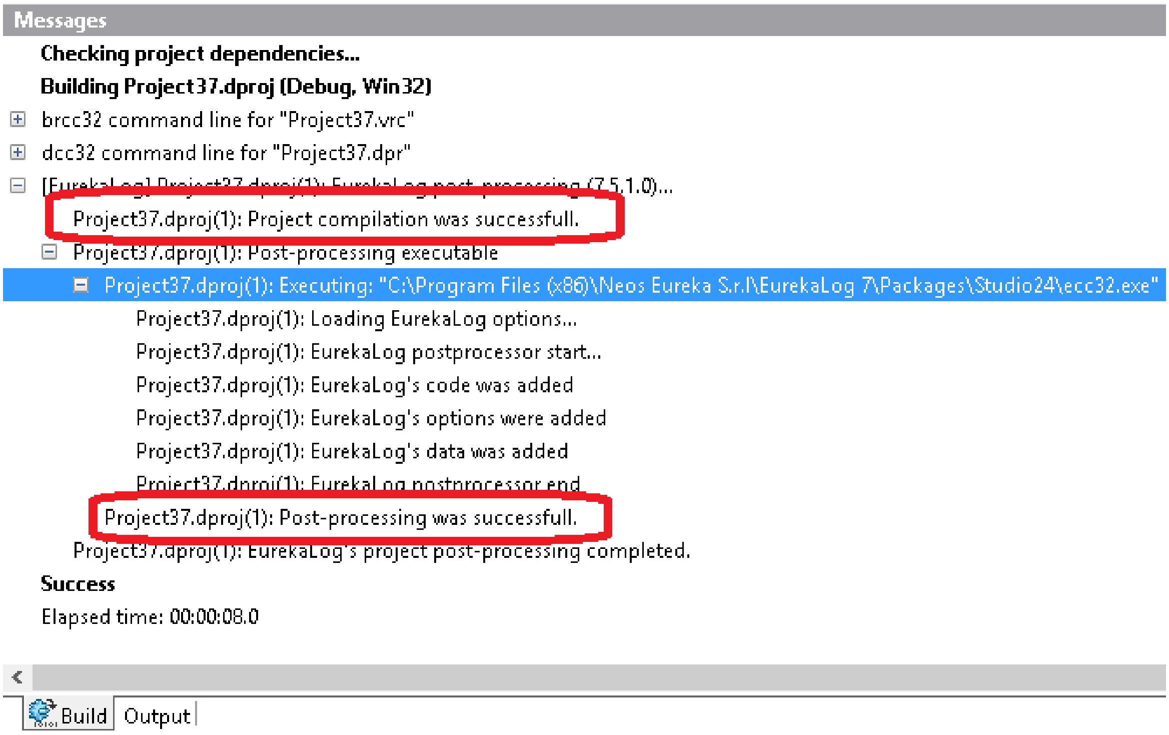
Current line:	0	Total lines:	54		
Hints:	0	Warnings:	0	Errors:	0

"EurekaLog" / "post-processing" stage during compilation. Notice "EurekaLog:" prefix on second line.



```
Messages
Checking project dependencies...
Building Project37.dproj (Debug, Win32)
brcc32 command line for "Project37.vrc"
dcc32 command line for "Project37.dpr"
[EurekaLog] Project37.dproj(1): EurekaLog post-processing (7.5.1.0)...
Success
Elapsed time: 00:00:08.0
```

"EurekaLog" / "post-processing" stage in IDE output
(use the following IDE main menu command: View / Messages)



```
Messages
  Checking project dependencies...
  Building Project37.dproj (Debug, Win32)
⊞ brcc32 command line for "Project37.vrc"
⊞ dcc32 command line for "Project37.dpr"
⊟ [EurekaLog] Project37.dproj(1): EurekaLog post-processing (7.5.1.0)...
    Project37.dproj(1): Project compilation was successfull.
  ⊟ Project37.dproj(1): Post-processing executable
    ⊟ Project37.dproj(1): Executing: "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio24\ecc32.exe"
        Project37.dproj(1): Loading EurekaLog options...
        Project37.dproj(1): EurekaLog postprocessor start...
        Project37.dproj(1): EurekaLog's code was added
        Project37.dproj(1): EurekaLog's options were added
        Project37.dproj(1): EurekaLog's data was added
        Project37.dproj(1): EurekaLog postprocessor end
      Project37.dproj(1): Post-processing was successfull.
    Project37.dproj(1): EurekaLog's project post-processing completed.
  Success
  Elapsed time: 00:00:08.0
```

Same window expanded - this is output from a normal successful compilation

Don't have EurekaLog IDE expert installed? 596

1. If you compile your application inside IDE - then it is the simplest case: because **you don't need to do anything**. You just enable EurekaLog for your project 33 and compile it.

2. However, if you're compiling your project outside IDE (say, you're using command-line compilation, build server or build script) or you don't have EurekaLog IDE expert installed (don't want to use it) - then you need to **post-process your compiled project manually** (see this article for more details 423).

Post-processing includes injecting additional information into your compiled executable module: EurekaLog's options and debug information 40.

See also:

- EurekaLog post-process compilers 423
- Reconfiguring EurekaLog for manual control 451
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- FinalBuilder 431
- EurekaLog compiler's command-line parameters 432
- Using EurekaLog with Delphi Personal/Turbo/Starter editions 23
- Working with EurekaLog configurations 439
- Configuring project for leaks detection 508

11.2.1 EurekaLog post-process compilers

Note: this article is part of explaining compilation outside of IDE 421.

EurekaLog provides two command-line compilers which performs post-processing 421: **ecc32.exe** for Delphi/RAD Studio and **emake.exe** for C++ Builder. These files are located in the same directory as EurekaLog packages. I.e. like '`C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe`' (this is just example).

Notes:

- Actually, **ecc32.exe** is exactly the same as **emake.exe** (it is the same file, but with different file name). Modern EurekaLog versions use the same file as both **ecc32.exe** and **emake.exe** for all IDE versions and personalities.
- Additionally, **ecc32speed.exe** is also provided. **ecc32speed.exe** is the same as **ecc32.exe**, except it does not contain debugging code, which allows it to execute slightly faster. EurekaLog uses **ecc32speed.exe** by default (from IDE). If debug output / verbose options are enabled, then EurekaLog will use **ecc32.exe**. There is no difference in calling **ecc32.exe** and **ecc32speed.exe**.

These new compilers integrate the standard **dcc32.exe/make.exe** features plus the EurekaLog features, so you can now just use the new **ecc32.exe/emake.exe** command-line compiler instead of the standard **dcc32.exe/make.exe** compiler. By using the EurekaLog command-line compiler - you can now compile your projects just like when you used the standard compiler, now adding to your projects the new EurekaLog features.

When **ecc32.exe/emake.exe/ecc32speed.exe** is called - it will compile your project and then add the EurekaLog settings and debug information to the application. Any parameters you pass to **ecc32.exe/emake.exe** are passed onto **dcc32.exe/make.exe** as in a normal compilation. The EurekaLog options are taken from the standard project options file 440.

Note: the **emake.exe** command-line compiler doesn't support ".bpr"/".bdsproj"/".cbproj" files (it only supports ".mak" files). To generate the ".mak" file from a ".bpr" you can use the **bpr2mak.exe** command-line program. See also: post-processing without compilation 426.

So **ecc32.exe/emake.exe/ecc32speed.exe** can be used just like **dcc32.exe/make.exe** and you can find plenty of information on these standard parameters in the Delphi help file. There is a minimal standard options 424 which have to be set for EurekaLog to function properly.

- If **ecc32.exe/emake.exe/ecc32speed.exe** is used without any EurekaLog custom parameters it will simply call the standard compiler (**dcc32.exe** or **make.exe**) and after that it will alter the compiled file by adding the EurekaLog options and debug data.
- If **ecc32.exe/emake.exe/ecc32speed.exe** is used with EurekaLog's **--el_prepare** or **--el_alter_exe** custom parameter (see here 426) it will **not** call the call the standard compiler (**dcc32.exe** or **make.exe**), it will only alter (project or already compiled) file by adding the EurekaLog options and debug data.

Additionally to standard parameters, **ecc32.exe/emake.exe/ecc32speed.exe** supports custom EurekaLog parameters 432 for customizing post-processing.

See also:

- Compiling your project with EurekaLog 421
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- FinalBuilder 431
- EurekaLog compiler's command-line parameters 432

11.2.2 Minimum parameters needed

Note: this article is part of explaining compilation outside of IDE 421.

If you're using build tool (such as IDE, FinalBuilder, etc.) to compile your project, then your project already have all required parameters for EurekaLog. See this article 58 for more information about required parameters and optimal setup.

However, if you're are compiling your project manually - then there is a minimum set of flags needed to get EurekaLog to work. You also need to tell the compiler where EurekaLog files are - so files can be included as part of the compiling process.

Absolute minimum is:

- Generation of debug information must be enabled (`-$D+`);
- Generation of map file must be enabled - "Detailed" is recommended choice (`-GD`);
- Path to EurekaLog's dcu/obj files must be specified (`-U<`*`path`*`>`);
- EurekaLog's conditional symbols must be set (`-D<`*`symbols`*`>`);

For example:

> **Example**
>
> "-UC:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Release\Studio16" -GD -$D+ -DEUREKALOG;EUREKALOG_VER7

Of course, you have to adjust:

- platform name (`Win32`)
- profile name (`Release`)
- IDE version name (`Studio16`)

The command line for **dcc32.exe** would looks like this:

> **Example**
>
> dcc32 "*your-project*" "-UC:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Release\Studio16" -GD -$D+ -DEUREKALOG;EUREKALOG_VER7

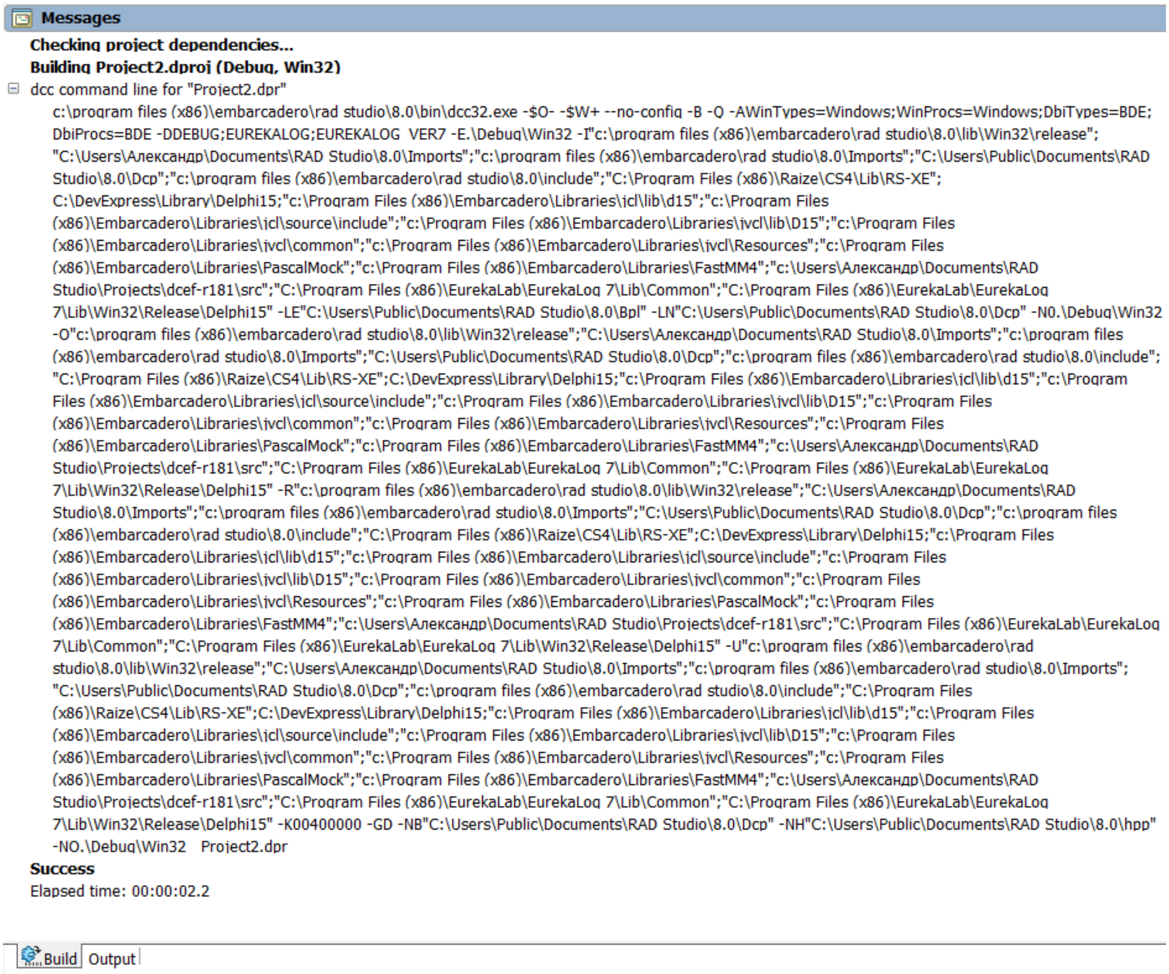
You may need to add more options 58 for your specific case. Please note that minimum options may be not enough for your needs. For example, you may want to have more debug options enabled (such as using debug version of system units, range-checking, etc.). Please see this article 58 for more information about recommended setup.

The command line for **ecc32.exe** would looks like this:

> **Example**
>
> ecc32 "*your-project*" "-UC:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Release\Delphi16" -GD -$D+ -DEUREKALOG;EUREKALOG_VER7 --el_config="*your-options*.eof"

You may need to add more options 432 for your specific case.

Maybe the simplest way to get a proper command-line for building your project is to install EurekaLog IDE expert (if it's not installed already), allow it to handle your project, compile your project and take a look at actual options for **dcc32.exe**/**make.exe**. You can do this via **View / Messages** command, switching to **Build** page and expanding **dcc command line** branch:



```
Messages
  Checking project dependencies...
  Building Project2.dproj (Debug, Win32)
⊟ dcc command line for "Project2.dpr"
    c:\program files (x86)\embarcadero\rad studio\8.0\bin\dcc32.exe -$O- -$W+ --no-config -B -Q -AWinTypes=Windows;WinProcs=Windows;DbiTypes=BDE;
    DbiProcs=BDE -DDEBUG;EUREKALOG;EUREKALOG  VER7 -E.\Debug\Win32 -I"c:\program files (x86)\embarcadero\rad studio\8.0\lib\Win32\release";
    "C:\Users\Александр\Documents\RAD Studio\8.0\Imports";"c:\program files (x86)\embarcadero\rad studio\8.0\Imports";"C:\Users\Public\Documents\RAD
    Studio\8.0\Dcp";"c:\program files (x86)\embarcadero\rad studio\8.0\include";"C:\Program Files (x86)\Raize\CS4\Lib\RS-XE";
    C:\DevExpress\Library\Delphi15;"c:\Program Files (x86)\Embarcadero\Libraries\jcl\lib\d15";"c:\Program Files
    (x86)\Embarcadero\Libraries\jcl\source\include";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\lib\D15";"c:\Program Files
    (x86)\Embarcadero\Libraries\jvcl\common";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\Resources";"c:\Program Files
    (x86)\Embarcadero\Libraries\PascalMock";"c:\Program Files (x86)\Embarcadero\Libraries\FastMM4";"c:\Users\Александр\Documents\RAD
    Studio\Projects\dcef-r181\src";"C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common";"C:\Program Files (x86)\EurekaLab\EurekaLog
    7\Lib\Win32\Release\Delphi15" -LE"C:\Users\Public\Documents\RAD Studio\8.0\Bpl" -LN"C:\Users\Public\Documents\RAD Studio\8.0\Dcp" -N0.\Debug\Win32
    -O"c:\program files (x86)\embarcadero\rad studio\8.0\lib\Win32\release";"C:\Users\Александр\Documents\RAD Studio\8.0\Imports";"c:\program files
    (x86)\embarcadero\rad studio\8.0\Imports";"C:\Users\Public\Documents\RAD Studio\8.0\Dcp";"c:\program files (x86)\embarcadero\rad studio\8.0\include";
    "C:\Program Files (x86)\Raize\CS4\Lib\RS-XE";C:\DevExpress\Library\Delphi15;"c:\Program Files (x86)\Embarcadero\Libraries\jcl\lib\d15";"c:\Program
    Files (x86)\Embarcadero\Libraries\jcl\source\include";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\lib\D15";"c:\Program Files
    (x86)\Embarcadero\Libraries\jvcl\common";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\Resources";"c:\Program Files
    (x86)\Embarcadero\Libraries\PascalMock";"c:\Program Files (x86)\Embarcadero\Libraries\FastMM4";"c:\Users\Александр\Documents\RAD
    Studio\Projects\dcef-r181\src";"C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common";"C:\Program Files (x86)\EurekaLab\EurekaLog
    7\Lib\Win32\Release\Delphi15" -R"c:\program files (x86)\embarcadero\rad studio\8.0\lib\Win32\release";"C:\Users\Александр\Documents\RAD
    Studio\8.0\Imports";"c:\program files (x86)\embarcadero\rad studio\8.0\Imports";"C:\Users\Public\Documents\RAD Studio\8.0\Dcp";"c:\program files
    (x86)\embarcadero\rad studio\8.0\include";"C:\Program Files (x86)\Raize\CS4\Lib\RS-XE";C:\DevExpress\Library\Delphi15;"c:\Program Files
    (x86)\Embarcadero\Libraries\jcl\lib\d15";"c:\Program Files (x86)\Embarcadero\Libraries\jcl\source\include";"c:\Program Files
    (x86)\Embarcadero\Libraries\jvcl\lib\D15";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\common";"c:\Program Files
    (x86)\Embarcadero\Libraries\jvcl\Resources";"c:\Program Files (x86)\Embarcadero\Libraries\PascalMock";"c:\Program Files
    (x86)\Embarcadero\Libraries\FastMM4";"c:\Users\Александр\Documents\RAD Studio\Projects\dcef-r181\src";"C:\Program Files (x86)\EurekaLab\EurekaLog
    7\Lib\Common";"C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Win32\Release\Delphi15" -U"c:\program files (x86)\embarcadero\rad
    studio\8.0\lib\Win32\release";"C:\Users\Александр\Documents\RAD Studio\8.0\Imports";"c:\program files (x86)\embarcadero\rad studio\8.0\Imports";
    "C:\Users\Public\Documents\RAD Studio\8.0\Dcp";"c:\program files (x86)\embarcadero\rad studio\8.0\include";"C:\Program Files
    (x86)\Raize\CS4\Lib\RS-XE";C:\DevExpress\Library\Delphi15;"c:\Program Files (x86)\Embarcadero\Libraries\jcl\lib\d15";"c:\Program Files
    (x86)\Embarcadero\Libraries\jcl\source\include";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\lib\D15";"c:\Program Files
    (x86)\Embarcadero\Libraries\jvcl\common";"c:\Program Files (x86)\Embarcadero\Libraries\jvcl\Resources";"c:\Program Files
    (x86)\Embarcadero\Libraries\PascalMock";"c:\Program Files (x86)\Embarcadero\Libraries\FastMM4";"c:\Users\Александр\Documents\RAD
    Studio\Projects\dcef-r181\src";"C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common";"C:\Program Files (x86)\EurekaLab\EurekaLog
    7\Lib\Win32\Release\Delphi15" -K00400000 -GD -NB"C:\Users\Public\Documents\RAD Studio\8.0\Dcp" -NH"C:\Users\Public\Documents\RAD Studio\8.0\hpp"
    -NO.\Debug\Win32   Project2.dpr
  Success
  Elapsed time: 00:00:02.2
```

A typical command line to build your project without using MS-Build

Note: this feature may be not available in your Delphi version.

If you're using a build script, then you already have this kind of "scary" command line - then all you need to do is to either change **dcc32.exe** to **ecc32.exe**, **make.exe** to **emake.exe** OR to add additional options as outlined in example above.

If you don't have this "scary" command-line yet - you may want someone else to take care about this. This "someone" can be IDE, build server/tool (like FinalBuilder). And in this case you need to post-process your file 426 **without compilation** 426 (since compilation along with "scary command-lines" will already be performed by other tool)...

See also:

- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Post-processing without (re)compilation 426
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- FinalBuilder 431
- EurekaLog compiler's command-line parameters 432

11.2.3 Post-processing without (re)compilation

Note: this article is part of explaining compilation outside of IDE 421.

This can be useful in situations where 3rd party build tools are not able to directly run ecc32.exe/emake.exe 423 (or you just don't want to build "scary" command-line 424). So, you're using "someone" to build your project as usual - this can be IDE, build server/tool (like FinalBuilder). Just don't forget to set mandatory options 58 for your project.

Now you need to post-process 421 your already compiled project without recompiling it. To achieve this we just need to use another EurekaLog command line parameter: --el_alter_exe 432.

Here is how the command line would look given that our application has already been compiled and we just want to add EurekaLog's features:

Example (Delphi 4-7)

```
ecc32 --el_alter_exe=Project1.dpr
```

Example (Delphi 2005-2006)

```
ecc32 --el_alter_exe=Project1.bdsproj
```

Example (Delphi 2007+)

```
ecc32 --el_alter_exe=Project1.dproj
```

Example (Delphi 2007+ with performance boost)

```
ecc32speed --el_alter_exe=Project1.dproj
```

Example (C++ Builder 5-6)

```
emake --el_alter_exe=Project1.bpr
```

Example (C++ Builder 2006)

```
ecc32 --el_alter_exe=Project1.bdsproj --el_mode=Builder
```

Example (C++ Builder 2007+)

```
ecc32 --el_alter_exe=Project1.cbproj --el_mode=Builder
```

Note: EurekaLog is able to recognize both project files (such as .dpr) and project configuration files (such as .dproj). It's recommended to specify project configuration files (such as .dproj) when possible.

If you want to specify the complied application's file name use:

Example

```
ecc32 --el_alter_exe=Project1.dpr;.\Debug\Win32\Project1.exe
```

(replace "ecc32" -> "emake" for C++ Builder, replace ".dpr" extension if needed - as in the previous example above)

You may also need to specify absolute or relative paths to files in some cases.

Important Note: post-processing will inject information into already compiled executable. It can not recompile or change the project. If you are switching between EurekaLog configurations - your .dpr source file may need to be changed (e.g. to change included EurekaLog units in uses clause). To achieve this we just need to use another EurekaLog command line parameter: --el_prepare 432.

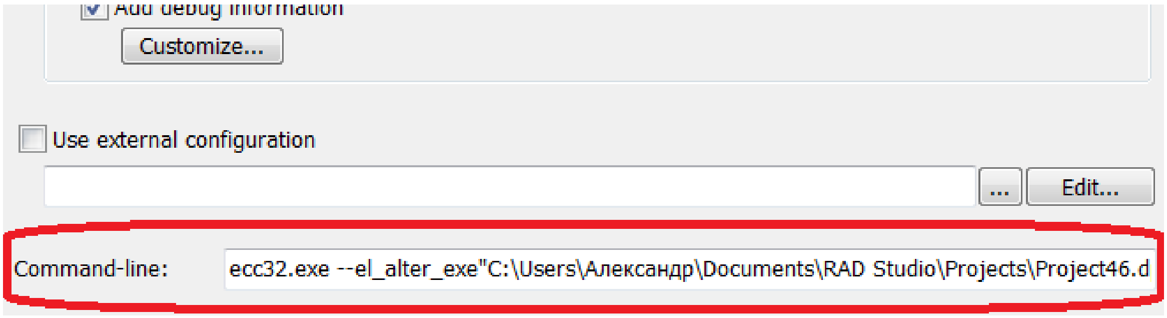
A typical batch file to build your project with EurekaLog may look like this:

Example

```
@echo off

REM Step 1: [Pre-Build] adjust .dpr source before compilation

ecc32 --el_prepare=Project1.dproj --el_config=Project1_Debug.eof

REM Step 2: compile your project

call "C:\Program Files (x86)\Embarcadero\RAD Studio\9.0\bin
\rsvars.bat"
msbuild Project1.dproj /t:Build /p:config=Debug;platform=Win32

REM Step 3: [Post-Build] post-process your project - inject options and/
or debug information

ecc32 --el_alter_exe=Project1.dproj;.\Debug\Win32\Project1.exe --
el_config=Project1_Debug.eof
```

`--el_config` switch is optional.

Note: EurekaLog shows command-line to post-processing your project for your convenience at General tab 234 in EurekaLog project options 225:



☑ Add debug information

□ Use external configuration

Command-line: ecc32.exe --el_alter_exe"C:\Users\Александр\Documents\RAD Studio\Projects\Project46.d

Ready to use command-line in EurekaLog project options

This edit is read-only. Just select all text, copy it into Windows clipboard and insert in place where you want to invoke EurekaLog's post-processing.

You can also use ecc32 to post-process executables compiled by non-Embarcadero compilers (such as Microsoft Visual Studio). You can use `NUL` as project file name and supply --el_config argument to specify location of EurekaLog options (.eof file). You can create .eof file by exporting EurekaLog options from any project or you can use standalone Settings Editor tool 617. You would need to add --el_source argument to indicate location of debug information (which is stored in PDB for Visual Studio).

Example

```
ecc32 --el_alter_exe=NUL;.\MS\VC\MSSample.dll --el_config=.\MS\VC
\MSSample.eof --el_source=PDB
```

See also:

- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Minimum parameters needed 424
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- FinalBuilder 431
- Reconfiguring EurekaLog for manual control 451

- EurekaLog compiler's command-line parameters 432

11.2.4 Using EurekaLog with automated builds

Note: this article is part of explaining compilation outside of IDE 421.

Direct calling of **ecc32.exe/emake.exe/ecc32speed.exe** rather than using standard EurekaLog expert is often used in automated build scenarios. Many EurekaLog's users setup a build server, where they can build working project automatically without calling IDE. Usually it involves usage some sort of make program (tool) or .bat/.cmd-files. In all those cases compilation is performed by calling **dcc32.exe/make.exe** directly. The EurekaLog expert is not used, so it has no chances in adding "EurekaLog's magic" to application.

In those cases you need to manually add a call to **ecc32.exe/emake.exe/ecc32speed.exe**. There are two options:

- [Script] If you have control over name of compiler ("dcc32.exe"/"make.exe") - then just replace it to "ecc32.exe"/"emake.exe" 423;
- [Tool] If you don't have control over name of "dcc32.exe"/"make.exe" - then you need to insert additional calls of ecc32.exe/emake.exe 426 with --el_prepare/--el_alter_exe parameters 432:
  - o [Delphi 2006 and below] You need to insert a call to **ecc32.exe/emake.exe/ecc32speed.exe** before and after project compilation 426;
  - o [Delphi 2007 and above] You can insert a call to **ecc32.exe/emake.exe/ecc32speed.exe** to project's pre- and post-build events 429.

See also:

- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Delphi 2007+ 429
- FinalBuilder 431
- Reconfiguring EurekaLog for manual control 451
- EurekaLog compiler's command-line parameters 432

11.2.5 Delphi 2007+

Note: this article is part of explaining compilation outside of IDE 421.

Using **ecc32.exe/emake.exe/ecc32speed.exe** with Delphi 2007+ is very easy. That's because Delphi 2007+ uses MS-Build tool and have pre/post-build commands 426 - so calling of **ecc32.exe/emake.exe/ecc32speed.exe** can be performed automatically during build.

Note: do not confuse IDE build events and EurekaLog build events 351. This article uses only IDE build events.

The simplest way to use **ecc32.exe/emake.exe** in Delphi 2007+ is to add a call to **ecc32.exe/emake.exe/ecc32speed.exe** to Pre-Build and Post-Build events:

Delphi command-line compiler

Pre-Build Event:
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe" "--el_prepare=$(PROJECTPATH)" "--el_profile=$(Config)"

Post-Build Event:
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe" "--el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)" "--el_profile=$(Config)"

C++Builder command-line compiler (RAD Studio)

```
Post-Build Event:
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages
\Studio16\ecc32.exe" "--
el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)" "--el_profile=$(Config)"
--el_mode=Builder
```

C++Builder command-line compiler

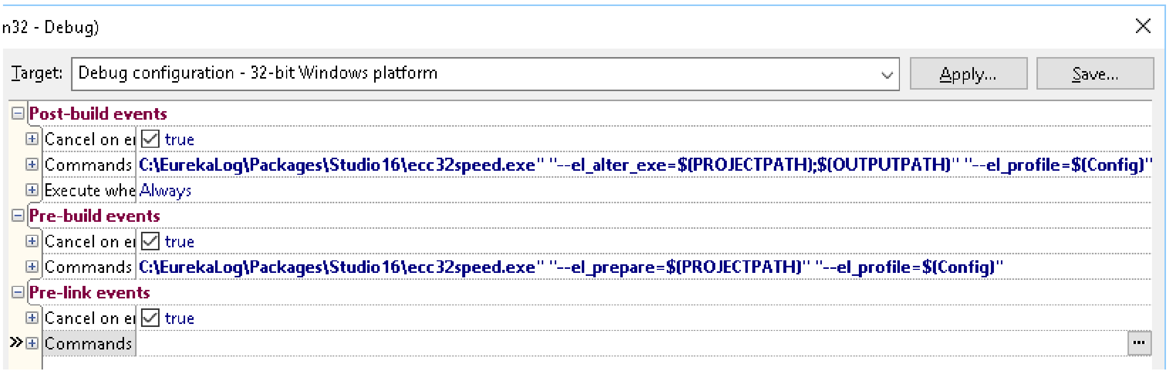
```
Post-Build Event:
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages
\Builder6\emake.exe" "--
el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)" "--el_profile=$(Config)"
```

(replace `Studio16` with real name of your IDE 604)

Important Note: "`Config`" variable may be not available in your IDE. In this case - please, remove `--el_profile` switch from the command line.

See Post-processing without (re)compilation 426 article for more information about used command-line switches.

For example:



n32 - Debug)

Target: Debug configuration - 32-bit Windows platform

Post-build events
- Cancel on er ☑ true
- Commands C:\EurekaLog\Packages\Studio16\ecc32speed.exe" "--el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)" "--el_profile=$(Config)"
- Execute whe Always

Pre-build events
- Cancel on er ☑ true
- Commands C:\EurekaLog\Packages\Studio16\ecc32speed.exe" "--el_prepare=$(PROJECTPATH)" "--el_profile=$(Config)"

Pre-link events
- Cancel on er ☑ true
- Commands

Adding a call to ecc32/emake/ecc32speed to your project

If you do this - you'll no longer need to perform any special actions to build your project. It doesn't matter how you build it: manually (msbuild Project1.dproj /t:Win32 / p:config=Debug), with build server/tool - the **ecc32.exe/emake.exe/ecc32speed.exe** will be invoked automatically (**as long as MS-Build is used**).

Notes:
- If you build your project *without* MS-Build (by calling **dcc32.exe/make.exe** directly) - then this option will have no effect. And you'll still need to call **ecc32.exe/emake.exe/ ecc32speed.exe** as explained here 423.
- Some build tools (such as FinalBuilder 431) may ignore MS-Muild settings, thus pre- and post-build events will not run. And you'll still need to call **ecc32.exe/emake.exe/ ecc32speed.exe** as explained here 423.
- You may want to disable EurekaLog IDE expert assist when using manual pre/post-build events. See this 448 and this 449 articles for more information on reconfiguring your project for manual control.

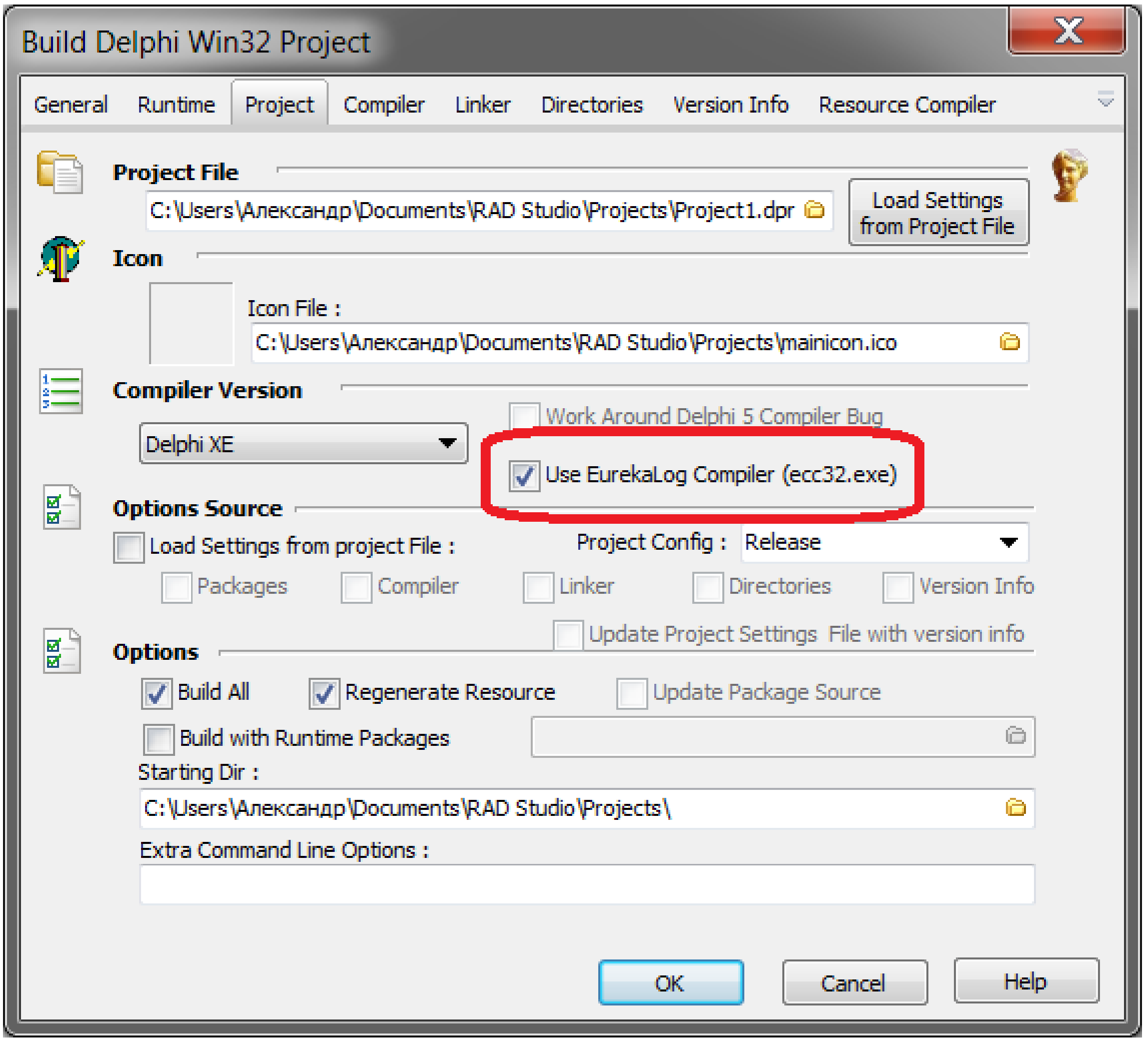
See also:
- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Reconfiguring EurekaLog for manual control 451
- Using EurekaLog with automated builds 429
- FinalBuilder 431

- EurekaLog compiler's command-line parameters 432
- Post-build events in EurekaLog options 351
- Different EurekaLog settings for "Debug" and "Release" profiles 448
- Read-only projects 449

11.2.6 FinalBuilder

Note: this article is part of explaining compilation outside of IDE 421.

Using **ecc32.exe**/**emake.exe** with FinalBuilder is very easy. That's because FinalBuilder has "Use EurekaLog compiler" option.



Build Delphi Win32 Project

General | Runtime | Project | Compiler | Linker | Directories | Version Info | Resource Compiler

Project File
C:\Users\Александр\Documents\RAD Studio\Projects\Project1.dpr
Load Settings from Project File

Icon
Icon File : C:\Users\Александр\Documents\RAD Studio\Projects\mainicon.ico

Compiler Version
Delphi XE
☐ Work Around Delphi 5 Compiler Bug
☑ Use EurekaLog Compiler (ecc32.exe)

Options Source
☐ Load Settings from project File :
Project Config : Release
☐ Packages ☐ Compiler ☐ Linker ☐ Directories ☐ Version Info
☐ Update Project Settings File with version info

Options
☑ Build All ☑ Regenerate Resource ☐ Update Package Source
☐ Build with Runtime Packages
Starting Dir : C:\Users\Александр\Documents\RAD Studio\Projects\
Extra Command Line Options :

OK | Cancel | Help

Build Delphi project options

All you need to do is to open properties of each "Build Delphi Win32 Project"/"Build C++ Builder Win32 Project" action and to enable "Use EurekaLog Compiler" checkbox. That's all.

Please, refer to FinalBuilder's help file for more information: **Actions Reference / Compilers / Embarcadero / Build Delphi Project**.

Notes:

- FinalBuilder will always build your project with your default EurekaLog configuration (unless you override configuration).
- Extra command line options are passed to ecc32.exe, so you may add any additional EurekaLog-specific switches 432. In other words, when "Use EurekaLog Compiler" option is enabled - FinalBuilder will call ecc32 instead of dcc32, --el_prepare/--el_alter_exe options are not used. Therefore, you can specify additional options for ecc32 (such as --

el_config, etc.; as well as arguments to dcc32) via FinalBuilder's "Extra Command Line Options" option.
- You *may* need to specify "Extra Command Line Options" option. This depends on your project's configuration. See Minimum parameters needed 424 to know about required parameters. Instead of manually specifying options - we recommend you to setup your project options as specified here 58 and mark "Load settings from project file", "Compiler" and "Linker" checkboxes on "Project" tab in FinalBuilder action's properties dialog. You may also uncheck "Load settings from project file" options, but setup compiler and linker options in the manner which is described in the above mentioned article.
- Alternatively, you may uncheck "Use EurekaLog Compiler" option, and add new action to your FinalBuilder script. Add "Exec program" action right after your "Build Delphi Win32 Project" action. Insert a call to ecc32/emake 426 with your custom options to this new "Exec program" action.
- FinalBuilder will not execute build events from MS-Build configuration. Therefore, if you have added a call to ecc32/emake/ecc32speed to your post-build event 429 - it will be ignored.

See also:
- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- EurekaLog compiler's command-line parameters 432

11.2.7 ecc32/emake command line options

Note: this article is part of explaining compilation outside of IDE 421.

EurekaLog provides **ecc32.exe** 421, 421 **emake.exe** 421, and 421 **ecc32speed.exe** 421 command-line compilers 421. These compilers pass all command line options to default **dcc32.exe** and **make.exe** compilers.

Additionally, these EurekaLog post-process compilers supports additional EurekaLog-related options, which allows you to customize EurekaLog's post-processing behaviour.

Notes:
- EurekaLog options switches are not passed to default **dcc32.exe** and **make.exe** compilers.
- EurekaLog options switches always start with "--el_" prefix.
- Do not forget to wrap arguments with spaces in double quotes (").
- EurekaLog is able to recognize both project files (such as .dpr) and project configuration files (such as .dproj). It's recommended to specify project configuration files (such as .dproj) when possible.
- EurekaLog can recognize arguments with and without "=". For example, the following arguments are equivalent to each other:

```
--el_config"Project1.eof"
--el_config=Project1.eof
"--el_config=Project1.eof"
```

EurekaLog IDE expert 221 calls **ecc32.exe/emake.exe/ecc32speed.exe** to post-process executables after compilation 423. Unlike command line, you have no direct control over calls to **ecc32.exe/emake.exe/ecc32speed.exe** when doing compilation from IDE. However, you still can use the custom EurekaLog arguments which are discussed below. Do the following to add any custom argument for **ecc32.exe/emake.exe/ecc32speed.exe** being invoked from IDE:
1. Open your project in IDE;
2. Go to the "Project" / "EurekaLog Options" menu;
3. Switch to the "Advanced" / "Custom/Manual" page;
4. Search for "ECC32AdditionalOptions" option. If such option not found (which is

default), you can add it to any location. Example of option:

```
ECC32AdditionalOptions="--el_compiler=dcc32speed.exe --
el_priority=$20"
```

Important note: Double-quotes in the example above is a part of the settings syntax for the "Custom/Manual" page 442 and not a part of the command-line arguments itself.

The above line is just example, you can alter it as you want to.

Notes:

- There is no "EMAKEAdditionalOptions" option. Use "ECC32AdditionalOptions" to add command-line options for both **ecc32.exe/ecc32speed.exe** and **emake.exe**
- "ECC32AdditionalOptions" option does not replace command-line, it merely adds new arguments. Therefore, you should be careful and do not specify arguments which are already in the command-line (e.g. `--al_alter_exe` and `--el_config`).

EurekaLog-specific options for **ecc32.exe/emake.exe/ecc32speed.exe** are:

1. --el-prepare/--el_alter_exe

These options instructs the EurekaLog compiler to not compile the project (via invoking dcc32/emake) and only add the EurekaLog options and debug data into already compiled application (you can also optionally specify the compiled application's filename).

1. By default EurekaLog's compiler compile your application AND post-processes it.
2. By specifying any of the options - you will disable compilation, so EurekaLog's compiler will perform only pre/post-processing.

Note: you must have compiled .exe/.dll/.bpl file and a corresponding .map file to use **--el_alter_exe** option.

Basically, **--el_prepare** switch acts as pre-build event, **--el_alter_exe** acts as post-build event:

1. **--el_prepare** switch will modify .dpr source file by adjusting list of EurekaLog units in uses clause. This is useful when you switch between different EurekaLog configurations. This switch will do nothing in most cases (nothing to change), thus it is optional, but recommended.
2. **--el_alter_exe** switch will inject EurekaLog options and compressed debug information into your executable. This is mandatory.

You should never specify both **--el_prepare** and **--el_alter_exe** in the same command line. Use either **--el_prepare** or **--el_alter_exe** or none.

These switches are useful when used with 3rd party tools that are not able to directly run the ecc32.exe command-line compiler. It's also useful if you don't want to bother with compiling your project by yourself (via direct call to dcc32/make) and assign this task to someone else (such as IDE, FinalBuilder, etc.).

Delphi Example (Delphi 4-7)

```
--el_prepare=ProjectFile.dpr
--el_alter_exe=ProjectFile.dpr[;ProjectFile.exe]
```

Delphi Example (Delphi 2005-2006)

```
--el_prepare=ProjectFile.bdsproj
--el_alter_exe=ProjectFile.bdsproj[;ProjectFile.exe]
```

Delphi Example (Delphi 2007+)

```
--el_prepare=ProjectFile.dproj
--el_alter_exe=ProjectFile.dproj[;.\Debug\Win32\ProjectFile.exe]
```

C++Builder Example

```
--el_alter_exe=MakeFile.mak
--el_alter_exe=ProjectFile.bpr[;ProjectFile.exe]
--el_alter_exe=ProjectFile.bdsproj[;ProjectFile.exe]
--el_alter_exe=ProjectFile.cbproj[;.\Debug\Win32\ProjectFile.exe]
```

Non-Embarcadero Example

```
--el_alter_exe=NUL;MSCppDll.dll
```

Path "`.\Debug\Win32\`" is only an example. Replace it with your value or just remove it (in case of `.exe` file being in the same folder as project file).

Part in [] is optional, but recommended. Of course, you should remove [] symbols for real-life examples (e.g. `--el_alter_exe=ProjectFile.cbproj;.\Debug\Win32\ProjectFile.exe`).

Note: you can specify `NUL` as project file name if the following conditions are satisfied:
1. You have specified compiled file name (optional part of **--el_alter_exe** switch);
2. You have specified config file name via **--el_config** switch (see below);
3. You have specified mode via **--el_mode** switch (see below).

You may optionally use **--el_source** switch (see below) for non-Embarcadero compilers. **--el_mode** switch have no effect for non-Embarcadero compilers.

See this article 496 to learn more about using `NUL` and **--el_source**.

2. --el_target

Specifies output file name. Optional. By default EurekaLog tries to get this from project's options (which is not always possible). It's not needed for **--el_alter_exe** switch, since output file name can be specified inside **--el_alter_exe** switch itself.

Example

```
--el_target=D:\Output\Bin\Project1.exe
```

Usually this switch is needed for projects with multiple configuration profiles, because EurekaLog may not know which profile to use.

Note: this option is ignored when **--el_prepare** or **--el_alter_exe** switches are used.

3. --el_profile

This option allows you to specify a profile name (build configuration). Profile name is used to select different configuration file. Usual names are 'Debug' and 'Release', but you are free to create any other custom configuration.

Profile name is used to select .eof file to load EurekaLog settings from. The base configuration is used by default (e.g. `Project1.eof`). Specifying non-empty profile name will load EurekaLog configuration from `Project1_`*`ProfileName`*`.eof` file instead.

Example

```
--el_profile=Debug
```

If there is no such .eof file - base configuration will be used.

Note: this option is ignored when **--el_config** switch is used.

4. --el_config

This option is used to compile your project with a different (alternative) EurekaLog options than options which are stored in the project itself.

Example

```
--el_config=Project1.eof
```

See also: how to get .eof file 227.

By default EurekaLog uses EurekaLog options from project configuration file (`.dof`, `.bdsproj`, `.dproj`, `.bpr`, `.cbproj`). It also respect external configuration option 443. You don't need to use this option if you want to compile your EurekaLog-enabled project (because your typical project configuration will be used automatically). You only need to use this option when your project does not contain EurekaLog configuration; or you want to override your settings.

See also:
- Using external configuration 443.
- EurekaLog post-process compilers 423.
- EurekaLog IDE expert 221.

5. --el_pid

Specifies Project ID (PID). Project ID is a unique GUID of the project. It is used by EurekaLog to identify projects. For example, for password storing (caching), BugID 240 creation, alias building in bug trackers, etc. Different projects are supposed to have different project IDs.

By default - project ID is extracted from .dproj, .bdsproj, .cproj files and then stored in base configuration (.eof file). You may want to specify project ID manually when using external configurations (e.g. using `--el_config` option), sharing configurations between projects, or if you simply have different project for essentially the same program.

Example

```
--el_pid={72C4CABF-E763-4606-89DE-5AA914D8F763}

--el_pid=72C4CABF-E763-4606-89DE-5AA914D8F763

--el_pid=72C4CABFE763460689DE5AA914D8F763
```

You may create a new GUID to use as project ID. Or you may copy GUID from .dproj/.bdsproj/.cproj file.

Note: this option is ignored when **--el_prepare** switch is used.

6. --el_mode

Switches between Delphi and C++ Builder project. Optional. You must add this switch only if you rename EurekaLog compiler's file. Possible values are "Delphi" or "Builder".

Delphi example

--el_mode=Delphi

C++ Builder example

--el_mode=Builder

Default is "Delphi mode" for **ecc32.exe/ecc32speed.exe** and "C++ Builder mode" for **emake.exe**.

Note: ecc32.exe and **emake.exe** is the same file.

7. --el_ide

Specifies alternative IDE version. IDE version affects which dcc32 is used to compile project, also how `.map` file is parsed, and how unit names are encoded. You have to use correct IDE version. E.g. version 7 when you work with project compiled by Delphi 7.

Delphi 7 example

--el_ide=7

RAD Studio 10.1 Berlin example

--el_ide=24

Default version is determinated by **ecc32.exe/emake.exe/ecc32speed.exe** location. For example, **ecc32.exe** from `C:\Program Files\EurekaLog 7\Packages\Delphi7\` will use version 7, **ecc32.exe** from `C:\Program Files\EurekaLog 7\Packages\Studio14\` will use version 14. Therefore, you have to specify this option only if you copy **ecc32.exe/emake.exe/ecc32speed.exe** to non-standard folder, or when you want to compile/post-process project for another/different IDE.

Note: see IDE name mapping 604.

8. --el_compiler

This options instructs the EurekaLog to use alternative command-line compiler instead of default dcc32/make,

Example

--el_compiler=dcc32speed.exe

"--el_compiler=C:\Program Files\CodeGear\Delphi\7.0\Bin\dcc32speed.exe"

Notes:
- This option is ignored when **--el_prepare** or **--el_alter_exe** switches are used.
- This is not the same as **--el_mode** option (see above). You can't change dcc32.exe to make.exe with **--el_compiler** option only. If you change Delphi compiler to C++ Builder (or visa versa) - you should use **--el_mode** option. If you want to change Delphi compiler to alternative Delphi compiler (or C++ Builder compiler to alternative C++ Builder compiler) - you should use **--el_compiler** option. Also, these two options can be used together.

9. --el_source

This option specifies which source should be used to create EurekaLog debug information. Default is .map file which should be in Borland-compatible format. Microsoft .map files are not supported. C++ Builder mode (see --el_mode switch above) regenerates .map file from .tds file (or injected TD32 info) - to complete .map file with line numbers.

Possible values are "" (empty string, default - use project's preferences), "MAP" (use .map file, default), "TDS" (use TD32), "PDB" (use PDF file), or "DBG" (use DBG file).

Warning: non-default values are experimental.

See this article 496 to learn more about using --el_source.

Example

--el_source=MAP

--el_source=PDB

Note: this option is ignored when **--el_prepare** switch is used.

10. --el_injectjcl/--el_createjcl

These options will additionally inject JCL (JEDI Code Library) debug information directly into your executable (`--el_injectjcl`), or create an external `.jdbg` file (`--el_createjcl`). This option does not replace normal EurekaLog post-processing. I.e. JCL debug information is added over EurekaLog's own data.

JCL debug information is supported by modern IDE versions, thus it is very useful to add to your .BPL files (and any other DLL which may be loaded/used by IDE).

Example

--el_injectjcl

--el_createjcl

Note: these options are ignored when **--el_prepare** switch is used.

11. --el_createdbg

This option will create an additional `.dbg` file.

This option requires **map2dbg.exe** tool present in EurekaLog's `\Bin` folder. **You have manually download this tool and copy it into `\Bin` folder of EurekaLog!** Check the following locations:

- https://code.google.com/p/map2dbg/
- https://github.com/andremussche/map2dbg
- https://github.com/garethm/map2dbg

DBG is debug information format supported by Microsoft. You can use this option when you need to work on your executables with Microsoft tools.

Example

--el_createdbg

Notes:

- Only 32-bit executables are supported by this option.
- This option is ignored when **--el_prepare** switch is used.

12. --el_verbose/--el_verbose_no_logs

These option instructs the EurekaLog compiler to produce more messages for debug purposes. If you have some strange issues with EurekaLog compiler - try to use this option to see what's going wrong.

Usually you don't need this option, it is used only for troubleshooting.

Example

--el_verbose
--el_verbose_no_logs

EurekaLog compiler will also create a few debug files. You can safely delete them if they aren't needed. They aren't used by EurekaLog and are generated only for debugging purposes. Those files are:

- *your-project*_EL_debug.eof - contains a copy of injected options.
- *your-project*.map_EL - contains a copy of injected debug information.

`--el_verbose_no_logs` option will not create additional log files.

Important Note: This option is limited in **ecc32speed.exe**. We recommend to use **ecc32.exe**/**emake.exe** for this option.

13. --el_gui_error

Shows errors in message boxes instead of writing them to the console. Optional.

Usually you want to add this switch, if you run compiler from IDE or other GUI tool. You probably don't want to specify it, if you run compiler from console or automated build script.

Note: EurekaLog compiler will set exit code (ERRORLEVEL) to non-zero value on any failure. Your build script can diagnose for success/failure compiles by analyzing process return code.

14. --el_priority

This option specifies class of process priority for ECC32/EMAKE. You can use any valid numeric value for process priority.

Example

--el_priority=$20

--el_priority=128

15. --el_nologo

This option suppresses standard ECC32/EMAKE's logo message with version and copyright information. You can use this option to get cleaner console output (for example, to redirect and analyze output).

This option does not affect actual processing of your project.

16. --el_nostats

This option suppresses writing compilation statistical information (such as sizes, times, etc.). You can use this option to get cleaner console output (for example, to redirect and analyze output).

This option does not affect actual processing of your project.

Examples

Some examples of full command lines:

Example 1

```
ecc32speed --el_alter_exe=Project1.dproj;.\Debug\Win32\Project1.exe
```

Example 2

```
emake --el_mode=Delphi --el_alter_exe=Project1.dproj;.\Debug\Win32\Project1.exe
```

Example 3
(for using in post-build event 429)

```
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32speed.exe" "--el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)"
```

Example 4

```
ecc32 Project1.dpr "-UC:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Release\Delphi7" -GD -$D+ -DEUREKALOG;EUREKALOG_VER7 --el_config=Project1_Debug.eof --el_verbose --el_gui_error
```

See also:

- Compiling your project with EurekaLog 421
- EurekaLog post-process compilers 423
- Minimum parameters needed 424
- Post-processing without (re)compilation 426
- Reconfiguring EurekaLog for manual control 451
- Using EurekaLog with automated builds 429
- Delphi 2007+ 429
- FinalBuilder 431
- Using external configuration 443

11.3 Working with configurations

This series of articles explains working with EurekaLog configuration in details.

Default behavior

EurekaLog installs IDE expert 221. IDE expert is extension for your IDE, which adds EurekaLog's menu items to IDE. It also allows saving additional information (EurekaLog options) with your project. By default EurekaLog stores its per-project configuration in .eof file with same name as your project. See Storing EurekaLog options 440 for more information.

Note: previous EurekaLog versions stored EurekaLog options inside project file instead of .eof file.

External configurations

You can also store EurekaLog configuration in external files. External EurekaLog configuration files also have .eof file extension.

.eof files can be obtained:
- by using "Export" button in project options dialog 227;
- by saving options to new file in external settings editor;
- by manual creation and editing in any text editor;
- by copying another .eof file.

EurekaLog will use external configuration if:
- it is specified as external options file in EurekaLog project options 234
- it was passed via --el_config option 432 to command-line post-processor 423

See Using External configuration 443 for more information.

Default folder for saving .eof files (i.e. folder for "Export configuration" dialog) is %AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\ (e.g. like C:\Users\UserName\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Profiles\). Any .eof file placed in that location will appear as "custom" project type 363. .eof files outside of the above mentioned folder will not appear in "Project type" option. Apart from %AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\ folder, a typical places to store .eof files are folder or sub-folder of your project.

Manual configuration

You can also manage EurekaLog configuration manually. This can be done:
- by uninstalling IDE expert
- by using "DoNotTouch*XYZ*" options

EurekaLog supports the following options, which you can enter at Custom/Manual page 356:
- DoNotTouch=1 - disables all EurekaLog's assist for the project. It will behave as if IDE expert is not installed at all.
  - DoNotTouchCompilation=1 - disables post-processing. You have to invoke ecc32/emake tools manually 426.
  - DoNotTouchOptions=1 - disables changing project options. You have to manually configure your project 58.
  - DoNotTouchUnits=1 - disables inserting/deleting unit into project's uses clause. You have to include EurekaLog's code manually.
  - DoNotTouchPackages=1 - disables modification of run-time packages list.

See Different EurekaLog settings for 'Debug' and 'Release' profiles 448 and Read-only projects 449 for two practical examples of manual controlled configuration.

See also:
- Using external configuration 443
- Compiling your project with and without EurekaLog 445
- Different EurekaLog settings for 'Debug' and 'Release' profiles 448
- Read-only projects 449
- Sharing EurekaLog settings in project group 450
- Reconfiguring EurekaLog for manual control 451
- Compiling your project with EurekaLog 421
- Using EurekaLog with other software 514

11.3.1 Storing EurekaLog options

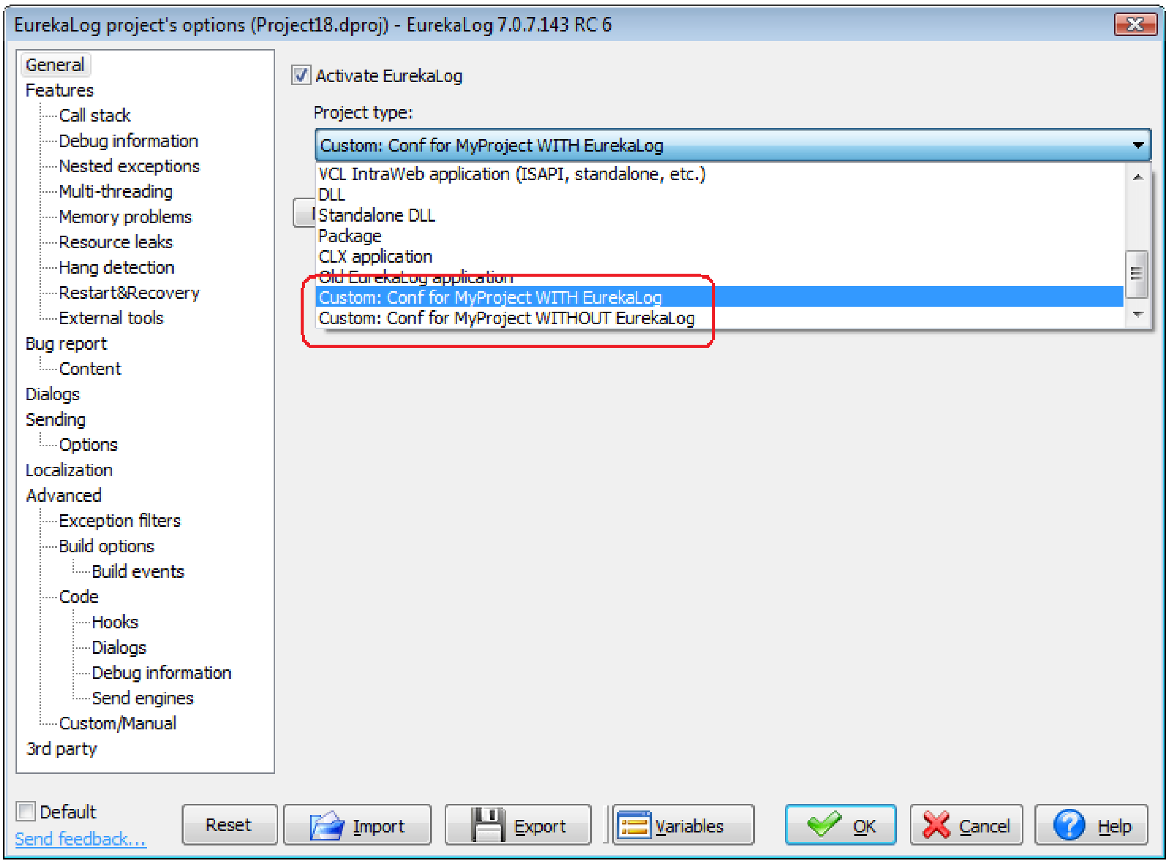
EurekaLog stores configuration in .eof files. .eof file is a UTF-8 encoded text ini-file, which can be edited in any text editor 442, or in specialized Settings Editor tool, included with EurekaLog.

EurekaLog stores **base** project configuration in file with same name as project file. For example, if you have Project1.dpr - then EurekaLog will save base configuration into Project1.eof file. Such file will be used by default in all cases (compilation from IDE, from command-line, etc.).

Additionally, you may manually create individual configurations for each profile (each build configuration). For example: Project1_Debug.eof. See Different EurekaLog settings for Debug and Release profiles 448 for more information.

Note: you may create additional (external) configurations 443.

Any .eof-file different from base or profile configuration is considered to be external configuration 443. Default folder for saving external .eof files (i.e. folder for "Import configuration" and "Export configuration" dialogs) is %AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\ (e.g. like C:\Users\UserName\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Profiles\). Any .eof file placed in that location will appear as "custom" project type 363.


EurekaLog project's options (Project18.dproj) - EurekaLog 7.0.7.143 RC 6

☑ Activate EurekaLog

Project type:

Custom: Conf for MyProject WITH EurekaLog

"Project type" option shows two .eof files

.eof files outside of the above mentioned folder will not appear in "Project type" option. For example, base configuration of your project will never appear as custom profile.

Apart from %AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\ folder, a typical places to store external .eof files are folder or sub-folder of your project.

Note: older EurekaLog versions stored EurekaLog options directly inside project options file:

DOF	Delphi 4-7
BDSPROJ	Delphi 2005-2006
DPROJ	Delphi 2007 and later

BPR	C++Builder 5-6
BDSPROJ	C++Builder 2006
CBPROJ	C++Builder 2007 and later

When you open such file in modern EurekaLog version - old options will be imported, saved into usual .eof file, and erased (technically, not erased: EurekaLog will simply not bother to save/write into project options file, and IDE will eventually overwrite non-standard blocks). You still can store settings in such files manually, and you can use Settings Editor tool to force-save settings into these files.

See also:

- Syntax for editing EurekaLog options 442
- Using external configuration 443
- Comping with EurekaLog 421
- Options / Custom/Manual page 356

11.3.1.1 EurekaLog options syntax

EurekaLog options are stored in text ini-like file 440. EurekaLog options are stored as set of "name=value" pairs. They are written under "[Exception Log]" section. You can edit options manually via text editor or Options / Custom/Manual page 356.

Notes:

- We do not recommend to manually edit EurekaLog settings, unless you need to add/change your own custom options, or change undocumented or hidden settings (e.g. setting which has no option in UI).
- EurekaLog stores full set of options when exporting options 227 to external configuration 443. EurekaLog stores only options with values different from defaults when saving base configuration 440.

A short example of .eof file contents:

```
[Exception Log]
EurekaLog Version=8000
Activate=1
atFixSafeCallException=1
atVCL=1
atWin32=1
DeleteMapAfterCompile=0
Encrypt Password=""
idEurekaLog=1
idEurekaLogDetailed=1
idMSClassic=1
idStepsToReproduce=1
InjectCode=1
InjectInfo=1
InjectOptions=1
loEnableMMDebugMode=1
ProjectID="{C2698C33-6841-47D6-980E-8692EE785B06}"
```

The following rules are applied:

- Each option should occupy single line only (line breaks inside text parameters are allowed via encoding - see below);
- Integer values are stored directly;
- Each option has form of 'Name=Value';
- Boolean values are stored as 0 (False) or 1 (True);
- Float values are stored as fixed-point values, decimal delimiter is always '.', e.g. '12.345';
- Enumeration values are stored as integers (not as text);
- String values are enclosed in double quotes ('"'). This allows you to store spaces and line breaks inside text parameters:
  - Empty strings may be stored as two double quotes ('""') or just as empty param ('');

- o '"' symbol should be escaped encoded as '\q';
- o Line breaks should be encoded as '\r\n';
- o '\' symbol should be escaped by doubling it (e.g. '\\');
- o '%' symbol should be escaped by doubling it (e.g. '%%');
- o While you can store non-ASCII symbols "as is" - we strongly recommend to encode them as '%U*hex-code*', for example, '%U044F' for unicode character #$44F (aka #1103, aka Cyrillic Small Letter Ya).

- Options are sorted for your convenience, but new options can be entered and placed in any order. There is no need to preserve sort order;
- Options with names started with "_" will not be saved into executable. Those are design-time only options, they are saved in project options, but not injected into final executable.
- There is no predefined list of allowed option names;
- We suggest to use 'Custom_' prefix for your own keys. EurekaLog will never have any option name, starting with 'Custom_'. Thus, your names will not collide with EurekaLog settings;
- You can retrieve any option at run-time via `CurrentEurekaModuleOptions.CustomField['Option-Name']`.

See also:
- Storing EurekaLog options 440
- Using external configuration 443
- Options / Custom/Manual page 356

11.3.2 Using external configuration

You may want to store configuration in separate .eof file for many reasons. For example, you may want to share common configuration between multiple projects. Another reason may be not having Delphi / C++ Builder project at all. For example, if you need EurekaLog configuration for Microsoft Visual Studio DLL 496.

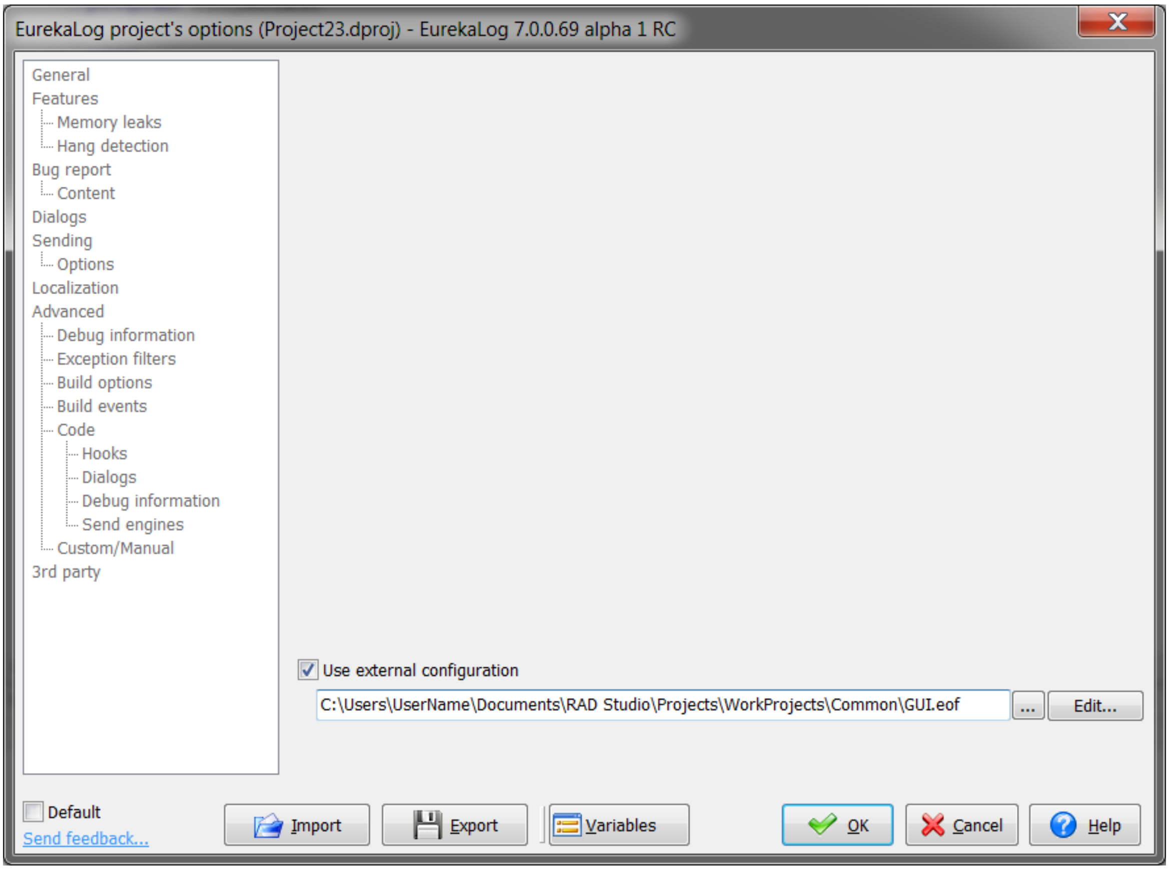
External configuration is stored in .eof files 440. These files are just renamed text ini-files 442.

.eof files can be obtained:
- by using "Export" button in project options dialog 227;
- by saving options to new file in external settings editor;
- by manual creation and editing in any text editor;
- by copying another .eof file.

Notes:
- Using the same configuration for different project *types* is usually not a good idea. For example, sharing configuration between packages, DLL, GUI and service applications is a bad idea. Create one configuration file for each type of projects in your project group (e.g. one configuration - for GUI; another configuration - for DLLs; and so on).
- Alternatively, you may want to copy configuration between projects. Use import / export functionality 227 in this case.
- Default location for saving .eof files (i.e. folder for "Export configuration" dialog) is `%AppData%\Neos Eureka S.r.l\EurekaLog\Profiles\` (e.g. like `C:\Users\`*`UserName`*`\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\Profiles\`).
- Any .eof file placed in default location will appear as "custom" project type 363. .eof files outside of the default location will not appear in "Project type" option.
- Apart from default location, a typical places to store .eof files are folder or sub-folder of your project.

1. Specifying external configuration file using IDE expert

You can specify external configuration file at "General" tab 234 in the EurekaLog project options dialog 225:



☑ Use external configuration

C:\Users\UserName\Documents\RAD Studio\Projects\WorkProjects\Common\GUI.eof ... Edit...

Specifying external configuration file in EurekaLog project options

Note: .eof file name can be arbitrary. It doesn't have to be the same name as the project; it doesn't have to be placed in the same folder.

2. Specifying external configuration file using command-line compiler

You can specify external configuration file via --el_config command-line option 432.

Note: ecc32/emake respects "use external configuration" option in the project (see above), so you can specify external configuration file with IDE expert - this option will be used by ecc32/emake. No need to use `--el_config` switch manually (but you still can use it, if you want to).

3. Using external configuration without IDE expert installed

When you don't have IDE expert installed - you can not edit EurekaLog configuration of the project from IDE. And no post-processing 421 will be performed. However, your *Project1*.eof file will still be untouched (not deleted). Therefore, you can edit it with external Settings Editor tool (or just plain text editor). And this file will automatically be used if you decide to make manual post-processing 426.

See also:

- Storing EurekaLog options 440
- Project options 234
- Command-line compilers 423
- Compiling your project with EurekaLog 421
- Compiling your project with and without EurekaLog 445
- Different EurekaLog settings for "Release" and "Debug" profiles 448
- Working with configurations 439

11.3.3 Compiling your project with and without EurekaLog

A common task for developers is compiling their projects with or without EurekaLog. For example, a developer may want to compile a project **with** EurekaLog for production (release), but compile the same project **without** EurekaLog for development (debug).

This use case have a good reasoning: EurekaLog is production diagnostic tool 68, which means it is designed to report about problems "post-morten", it also means that EurekaLog uses fast-enough approaches, as opposed to using heavy debugging code. Therefore, it is preferable to use debug tools (such as IDE debugger, debugging memory manager, OS's checked build, etc.) to locate issues while developing applications, and use EurekaLog to catch remaining issues "on the field".

And some developers use EurekaLog only for local debugging, so they want to enable EurekaLog for development, but disable it for production.

Important Note: This article will discuss the case when you want to have EurekaLog enabled in one profile, but not in another. If you want EurekaLog to be enabled for both Debug and Release profiles, but with different configuration - please, see Different EurekaLog settings for 'Debug' and 'Release' profiles 448 article instead.

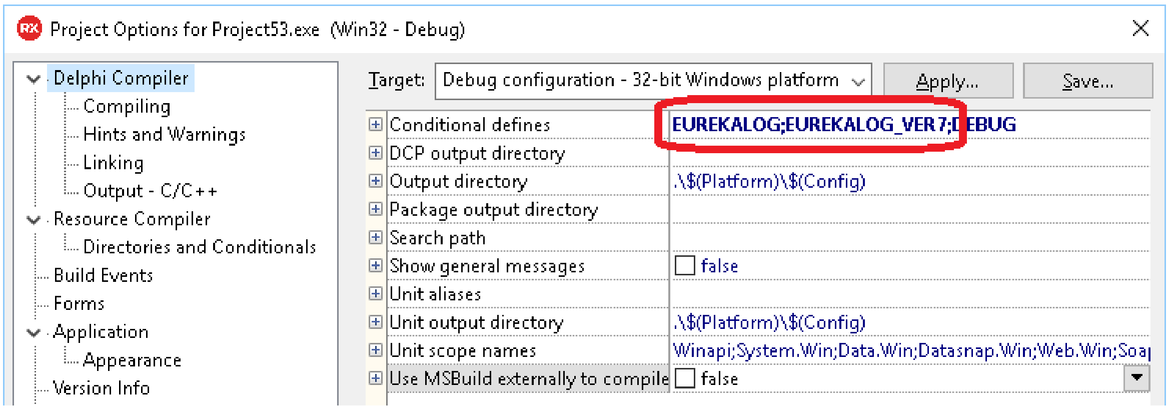
Anyway, the basic idea is that you (as developer) may want to compile your application with and without EurekaLog. And you also need to do this (i.e. switch configurations) regularly. There are several possible methods to do this.

1. (Correct) method #1: Define/Undefine EUREKALOG conditional symbol

EurekaLog units are included in your project as this:

```
program Project1;

uses
  {$IFDEF EurekaLog}
  EMemLeaks,
  EResLeaks,
  EDialogWinAPIMSClassic,
  EDialogWinAPIEurekaLogDetailed,
  EDialogWinAPIStepsToReproduce,
  EDebugExports,
  EFixSafeCallException,
  EMapWin32,
  EAppVCL,
  ExceptionLog7,
  {$ENDIF EurekaLog}
  your-units;
```

Where EUREKALOG symbol is defined in your project options:



Project Options for Project53.exe (Win32 - Debug)

- Delphi Compiler
  - Compiling
  - Hints and Warnings
  - Linking
  - Output - C/C++
- Resource Compiler
  - Directories and Conditionals
- Build Events
- Forms
- Application
  - Appearance
- Version Info

Target: Debug configuration - 32-bit Windows platform

Conditional defines	EUREKALOG;EUREKALOG_VER7;DEBUG
DCP output directory	
Output directory	.\$(Platform)\$(Config)
Package output directory	
Search path	
Show general messages	□ false
Unit aliases	
Unit output directory	.\$(Platform)\$(Config)
Unit scope names	Winapi;System.Win;Data.Win;Datasnap.Win;Web.Win;Soa
Use MSBuild externally to compile	□ false

EUREKALOG symbol is define in project's options

You may undefine (e.g. remove) EUREKALOG symbol from certain build configurations (profiles). Do not remove EUREKALOG_VER7 symbol, remove only EUREKALOG symbol.

Note: EurekaLog post-processing will first check if EUREKALOG symbol is present; if it is not defined - post-processing will be skipped.

2. (Wrong) method #2: Use "Activate EurekaLog" option

The first, an obvious (and a wrong) way to archive the above mentioned goal is to use "Activate EurekaLog" option 234. While this *may* work for many projects (basically - for those, which options were not changed much), this option will not get the desired behavior for all cases. For example, if you want to use EurekaLog in your project, but do not want to use EurekaLog's memory debugging features 250, then "Activate EurekaLog" option will not work for you: turning on "Activate EurekaLog" option will also turn on "Enable extended memory manager" option.

Long technical explanation

1. "Activate EurekaLog" option is **not a real option**. It is a meta-switch, which does nothing by itself, it just enables or disables *other* options (and among them is the "Enable extended memory manager" option). The primary goal of "Activate EurekaLog" switch is to enable EurekaLog and bring it to default working condition.

2. "Enable extended memory manager" option is designed to work *with* **AND** *without* EurekaLog. In other words, you can disable EurekaLog in your project and enable "Enable extended memory manager" option - and have EurekaLog's memory filter. This is a designed behaviour - to allow to have shared memory manager in an executable module which is designed to work with other EurekaLog-enabled executable module. An obvious default state for "Enable extended memory manager" option is **disabled**, i.e. in a brand new project "Enable extended memory manager" option will be in disabled state - because otherwise all projects will have EurekaLog memory filter on board. So, when you enable "Activate EurekaLog" option - it has to turn on the "Enable extended memory manager" option.

P.S. There are few additional optional features which can work with and without full EurekaLog in your project - see "Additional hooks" option on this page 352.

Therefore, while sometimes using "Activate EurekaLog" option may work for your project, there are cases when it will not work as you want it to. For such cases there are several other methods - listed below.

3. Method #3: Always compile with EurekaLog, but disable EurekaLog at run-time

The easiest and most simple method is just have the same executable for all cases. Compile your project with EurekaLog - as you would do it for production/release. Then add check like this at startup:

```
if ThisIsADeveloperMachine then
  SetEurekaLogState(False); // completely disable EurekaLog
```

where ThisIsADeveloperMachine is your custom function to determinate if you are running on developer machine or not. You may use IsDebuggerPresent function as ThisIsADeveloperMachine function. Or you may read registry for specific "magic" value.

You may also disable only part of EurekaLog - instead of disabling the entire EurekaLog. For example, you may turn off leaks registering (call EMemLeaks.DisableMemLeaksRegistering). Or you may turn off sending (set CurrentEurekaLogOptions.SenderClasses := esmNoSend). Etc.

If you are not satisfied with run-time customization (as it will still add performance penalty at compilation and startup) - then you have to really compile your application without EurekaLog.

4. (Workaround) method #4: Disable "Catch memory leaks" and use "Activate EurekaLog" option

Often a reason behind having "Enable extended memory manager" option turned off is performance considerations. If this is a case - then you can use the following method: turn on "Enable extended memory manager", turn off "Catch memory leaks" (and possible some other options on "Memory problems" page - but keep "Enable extended memory manager" turned on), and use "Activate EurekaLog" option to turn EurekaLog on/off. This way you don't need to have different settings for different profiles.

Long technical explanation
1. You get most performance hit from "Catch memory leaks" option, as EurekaLog has to build call stack for each memory allocation; while "Enable extended memory manager" option itself does not have major impact on performance.
2. It is strongly recommended to keep "Enable extended memory manager" option turned on when possible, as EurekaLog uses MM filter to track lifetime of exception objects. The worst case scenario would be: using Delphi 2007 or earlier, having "Enable extended memory manager" option turned off, having "Use low-level hooks" option 259 turned off - EurekaLog won't be able to track lifetime of exception objects in such configuration.

5. Method #5: Switch real options instead of switching "Activate EurekaLog" meta-switch

As mentioned above, the "Activate EurekaLog" option is a meta-switch: it turns on/off *other* options. Therefore, the problem is that "Activate EurekaLog" option may turn on/off some particular option that you want to remain untouched ("Enable extended memory manager" option is just one example, discussed above - method #4).

The obvious way around this behavior is to ignore "Activate EurekaLog" option and **turn on/off individual options**.

For example, instead of switching "Activate EurekaLog" option - try switching "Add EurekaLog's code", "Add module's options" and "Add debug information" options (expand advanced view on "General" page 234 to see these options).

6. Method #6: Use custom configurations

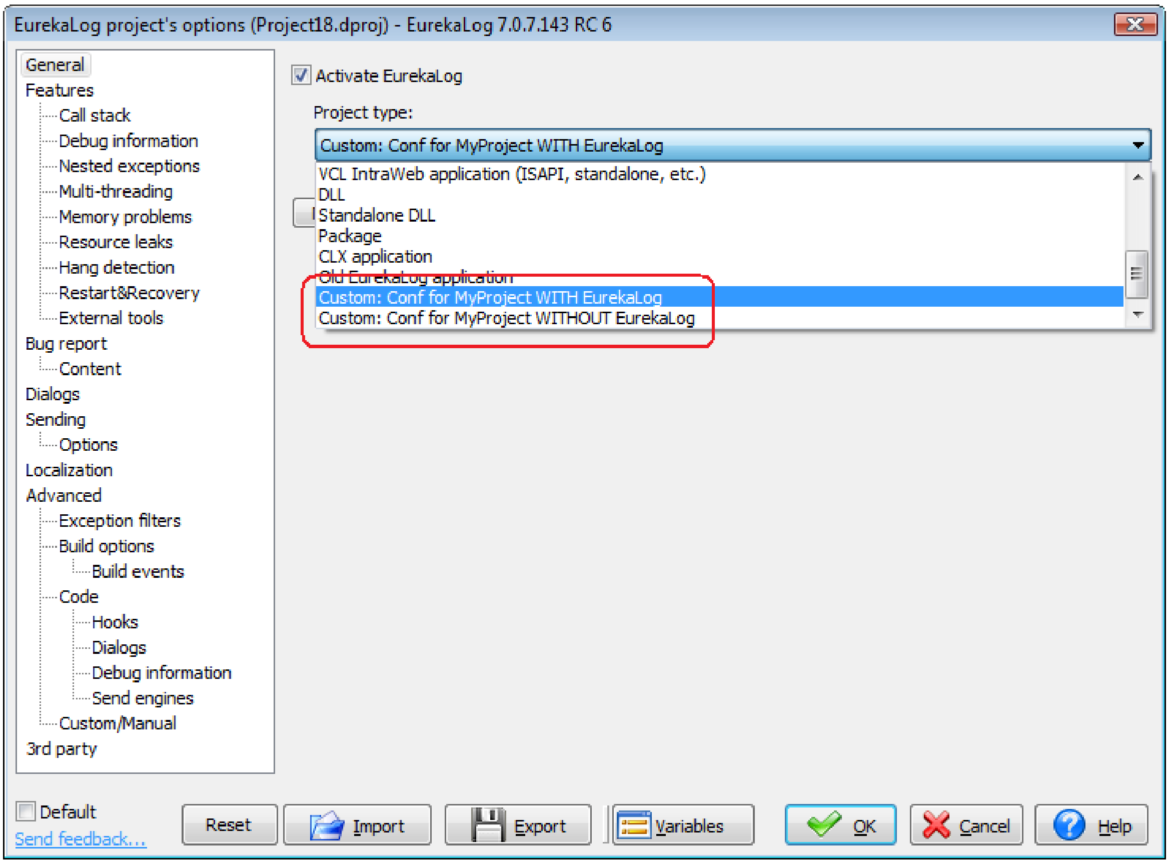
Switching individual options (method #4) may become rather inconvenient and prune to errors - especially when number of options to change is quite large.

There is a way to switch all options at once:

- Create configuration for your project with EurekaLog enabled;
- Export EurekaLog configuration in external .eof file, be sure to save it in the default folder (i.e. do not change folder in "Export" dialog), give it a descriptive name like "Configuration for MyProject with EurekaLog";

- Create configuration for your project without EurekaLog enabled;
- Export EurekaLog configuration in external .eof file, be sure to save it in the default folder (i.e. do not change folder in "Export" dialog), give it a descriptive name like "Configuration for MyProject without EurekaLog";

Now you can switch between two sets of options via "Application type" combo-box 234 - just in two clicks:


EurekaLog project's options (Project18.dproj) - EurekaLog 7.0.7.143 RC 6

☑ Activate EurekaLog

Project type: Custom: Conf for MyProject WITH EurekaLog

- VCL IntraWeb application (ISAPI, standalone, etc.)
- DLL
- Standalone DLL
- Package
- CLX application
- Old EurekaLog application
- Custom: Conf for MyProject WITH EurekaLog
- Custom: Conf for MyProject WITHOUT EurekaLog

Custom profiles in "Project type" combo-box

You can also use --el_config option 432 and define/undefine EUREKALOG conditional symbol.

7. Method #7: Use different compilation profiles

Alternative to method #6 (having several EurekaLog configurations for the same compilation/build profile) is method #7: have different EurekaLog configurations for different compilation/build profiles. This method is harder to setup and maintain, but it allows you to use build-in IDE's "profiles" feature to switch between different configurations. This method is described in a separate article 448.

See also:

- Compiling your project with EurekaLog 421
- Using external configuration 443
- Different EurekaLog settings for "Release" and "Debug" profiles 448
- Reconfiguring EurekaLog for manual control 451
- Read-only projects 449
- Working with configurations 439

11.3.4 Different EurekaLog settings for 'Debug' and 'Release' profiles

This article is supposed to answer on one common question asked by our customers. They want to use different EurekaLog settings for different compilation profiles. This article will explain how to do this.

Important note: This article will discuss the case when you want to have EurekaLog enabled in both Debug and Release profiles, but with different configuration. If you want EurekaLog to be enabled for Debug profile and be disabled for Release profile (or visa versa) - please, see Compiling your project with and without EurekaLog 445 article instead.

Well, first - unfortunately, there is no IDE solution for certain technical reasons. But this doesn't mean that you can't do this. You can't use automatic solution, but you can perfectly set all options manually.

For the purposes of this article we will use EurekaLog 7.6 and Delphi XE. The discussed features *may* be unavailable in older versions. Please, see Reconfiguring EurekaLog for manual control 451 article for an alternative method to set up profiles.

See also:

- Storing EurekaLog options 440
- ecc32/emake command line options 432
- Using external configuration 443
- Compiling your project with and without EurekaLog 445
- Reconfiguring EurekaLog for manual control 451
- Read-only projects 449
- Working with configurations 439

11.3.5 Read-only projects

Some developers set their project files (such as .dpr or .dproj) to read only. Read-only project files are used as indicators to version control software. However, putting your project to read-only will prevent EurekaLog from working properly - because EurekaLog needs to modify your project to include EurekaLog units and adjust options for debugging.

Step 1: initial setup of EurekaLog in your project

First, you have to configure EurekaLog in your project for the first time. You have to remove read only attribute from your project files to do this. Next, open your project, open EurekaLog project options and configure EurekaLog as you usually would do.

This step will set your project options for debugging, include EurekaLog units into your application, and store EurekaLog options with your project.

Step 2: [optional] disable EurekaLog's assist for project files

Next, you have to tell EurekaLog that it should not try to modify your project files. Usually EurekaLog will try to do this. If your project files are set as read only - EurekaLog will fail and ask you to make your files writable. However, since we already set up EurekaLog - there is no need to make any further modifications.

Note: EurekaLog will only try to modify your project files if it does not match EurekaLog configuration (for example, map file generation is disabled, or uses clause in DPR file does not include EurekaLog units). If there is no need to make changes - EurekaLog will not try to modify/write project files. Therefore, you may skip this step and use "file is read-only" errors as indication that configuration was changed and project needs to be adjusted.

You can instruct EurekaLog to skip project modifications this by enabling "DoNotTouchOptions", "DoNotTouchUnits", and "DoNotTouchPackages" options. Open EurekaLog project options (Project / EurekaLog options), go to "Advanced" / "Custom/ Manual" category, add the following lines to any location:

```
DoNotTouchOptions=1
```

```
DoNotTouchUnits=1
DoNotTouchPackages=1
```

Close dialog with OK button and save your project. Now EurekaLog will not try to modify project options or list of units.

Step 3: re-enable read-only status

Finally, it's time to enable your read-only attribute back for project files. Your project files can be read-only and EurekaLog will work. EurekaLog will be fully functional. EurekaLog will post-process your project on each compilation by using currently set options.

Non-EurekaLog read-only projects

If you want to work with read-only projects without using EurekaLog for them, but still have EurekaLog IDE expert installed - then you have to instruct EurekaLog to skip your non-EurekaLog projects (otherwise EurekaLog may try to remove itself from the project and fail cause project is read-only). To do this - create empty .eof text file with the same name as your project. For example, if you have `Project1.dpr` project - then you should create new `Project1.eof` file. Place the following lines to this .eof file (use Notepad to edit file; it's a text file):

```
[Exception Log]
DoNotTouch=1
```

Save changes. Done.

See also:
- Storing EurekaLog options 440
- Using external configuration 443
- Compiling your project with and without EurekaLog 445
- Different EurekaLog settings for "Debug" and "Release" profiles 448
- Reconfiguring EurekaLog for manual control 451
- Working with configurations 439

11.3.6 Sharing EurekaLog settings in project group

Sometimes you work on a project with large amount of Delphi projects grouped into project group. You may either want to share a common EurekaLog settings between projects or set EurekaLog settings for few projects simultaneously.

Generally speaking, blindly sharing EurekaLog settings between several different projects is not a good idea. Various projects may be of a different type: like VCL application, DLL, packages, etc. These projects must have slightly or significantly different EurekaLog settings.

Therefore, a better idea would be to put all of your projects into logical groups, each group should have exactly the same EurekaLog settings. For example, you may select a group of 1 project for main executable, a group for DLLs and a group for packages. In other words, your project group may have either one or several logical groups of projects.

Once you have decided which projects should have different EurekaLog settings - it is time to actually assign them. Unfortunately, IDE does not provide a way to edit options of multiple projects simultaneously. You only can edit one project at time. Therefore, you should use one of the following workarounds:

1. Option #1: Export / Import

1. You should configure EurekaLog for one particular project in a logical group.
2. Then you can use "Export" button 227 to export EurekaLog settings into .eof file.
3. Save this file somewhere outside of project sources.

4. Open each other project from the same logical group.
5. Import EurekaLog settings from .eof file saved on steps 2-3. Use "Import" button[227] for that.
6. Repeat steps 4-5 for each project in logical group.
7. Repeat steps 1-6 for each logical group. You may re-use saved configuration as base settings for step 1.
8. Done. You may delete .eof file, it is no longer needed.

This sequence will assign exactly the same EurekaLog settings for all projects in the same logical group. However, any further changes to EurekaLog settings in any of projects will not affect other projects in the same logical group. If you want to make changes in all projects in a logical group - then you must edit options in one project and then repeat steps 2-6.

2. Option #2: Using External Configuration

1. You should configure EurekaLog for one particular project in a logical group. Choose project carefully, as it will become a master-project for your group.
2. (Optional) Use "Export" button to save EurekaLog settings into .eof file.
3. Open each other project from the same logical group.
4. Check "Use external configuration" option[234] and select .eof file created on step 2 (an open dialog will be opened automatically; if not - use "..." button to manually show open dialog).
5. Repeat steps 3-4 for each project in logical group.
6. Repeat steps 1-5 for each logical group. You may import saved configuration as base settings for step 1. However, do not save all configurations into a single file. You must save each configuration into file corresponding to the project.
7. Done. Do not delete any .eof files - they are required.

This sequence will also assign exactly the same EurekaLog settings for all projects in the same logical group. However, now your settings are stored in the single .eof file rather than individual .eof files for each project. The advantage of this method is that you can edit EurekaLog options of master-project - and this will automatically "adjust" options in all projects from the same logical group.

If you want to make some particular project to be different from its logical group - then you can uncheck "Use external configuration" option, import .eof file from master-project, and adjust settings for this project.

Note: there is no inheritance support in EurekaLog's settings.

See also:

- Storing EurekaLog options[440]
- Using external configuration[443]
- Compiling your project with and without EurekaLog[445]
- Different EurekaLog settings for "Debug" and "Release" profiles[448]
- Reconfiguring EurekaLog for manual control[451]
- Working with configurations[439]

11.3.7 Reconfiguring EurekaLog for manual control

This article will discuss how you can control EurekaLog manually - without assistance of IDE expert[604]. This may be useful if you want to use EurekaLog without IDE expert installed, if you want consistent behavior between IDE and your build tool, or for any other reason.

Step 1: get working solution for single profile

First, we create a new VCL application and place a button to raise exception. Then go to Project/EurekaLog options, enable EurekaLog and specify type of this application (VCL Forms). You can also set other options as you desire. Now run the application and confirm it's working as expected.

Step 2: reconfigure project for manual control

A: Creating options
For the next step you should go to "Project" / "EurekaLog options" and use "Export" button to create .eof file. Place it in the same folder as your project (by default "Profiles" folder is suggested, switch to your project's folder). Name it as you like. For example: `Project1_Custom_Config.eof`.

B: Disable IDE assist
Now, don't close options dialog, but go to "Advanced" / "Custom/Manual" and add "`DoNotTouch=1`" line (without quotes) in any place (as new line). This will disable any assist for your project from IDE expert. Close settings and save your project.

Note: do not add "`DoNotTouch=1`" line to configuration created on step A above.

You can confirm if option is taking effect by disabling EurekaLog, saving your project and observing that there are no changes in your .dpr file - all units are still included even if no EurekaLog is enabled.

Note: EurekaLog also supports "`DoNotTouchCompilation`", "`DoNotTouchOptions`", "`DoNotTouchUnits`", and "`DoNotTouchPackages`" options for more precise control over options. Set "`DoNotTouch`" option to completely disable EurekaLog assist for the project; or set one or several "`DoNotTouch`*`XYZ`*" options. "`DoNotTouchCompilation`" option disables post-processing assist. Enable this option if you intend to invoke ecc32/emake manually. "`DoNotTouchOptions`" disables modification of project options and defines. Enable this option if you want to manage project options manually. "`DoNotTouchUnits`" disables modification of units in uses clause. Enable this option if you want to manually manage your units. "`DoNotTouchPackages`" option disables modifications in run-time packages list.

Alternatively, you may simply disable/remove EurekaLog IDE expert.

C: Setting up post-processing
Now, it's time to restore post-processing for your application 421. Go to "Project" / "Options" (not EurekaLog options) and look for build events options 429.

Add the following command as pre-build event which is invoked before compilation:

```
IF EXIST "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ec
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe"
"--el_prepare=$(PROJECTPATH)"
"--el_config=Project1_Custom_Config.eof"
```

Add the following command as post-build event which is invoked after successful compilation:

```
IF EXIST "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ec
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe"
"--el_alter_exe=$(PROJECTPATH);$(OUTPUTPATH)"
"--el_config=Project1_Custom_Config.eof"
```

Notes:

- Both command lines are broken into several lines for readability. Do not break into lines when entering command-line to pre/post-build events in IDE;
- Replace "`C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\`" folder with real EurekaLog installation path on your machine;
- Replace "`Studio16`" with corresponding name 604 for your IDE;
- Replace "`Project1`" with your real project name;
- You can find the IDE's `$(Config)` and `$(Platform)` variables useful. `$(Config)` will be replaced with build configuration name (profile) - such as Debug and Release. `$(Platform)` will be replaced with short name of the platform - such as `Win32`, `Win64`,

OSX. So you can have file like `Project1_Win32_Debug.eof` and use `"--el_config=Project1_$(Platform)_$(Config).eof"` switch.

So, the resulting command-line may look like this when run:

```
IF EXIST "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ec
"C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio16\ecc32.exe"
--el_alter_exe"C:\Projects\Project1.dproj;C:\Projects\Debug\Win32\Project1.exe"
"--el_config=Project1_Custom_Config.eof"
```

Note that this is an example of final command as it will be executed by IDE. You should NOT use this form of command (with already expanded variables) - please use the first example with `$(...)` variables.

D: Check that everything works
Now compile your project and run it. If you have done everything correctly - the result must be the same as on step 1: the correct EurekaLog-enabled application with expected behavior as set in external .eof file (even though the EurekaLog was disabled in project).

In case of any build errors - take a look at compiler output as shown in "Messages" window. It's docked at the bottom of IDE window by default. "Output" tab is near "Build" tab, which is active by default. If you don't see "Messages" window - use View/Messages command to show it, then switch to output window. The correct compilation will get you such messages:

```
Build started 2012.06.29 16:58:04.
__________________________________________________
Project "C:\Projects\Project1.dproj" (Build target(s)):
Target _PasCoreCompile:
     C:\program files (x86)\embarcadero\rad studio\8.0\bin\dcc32.exe //-- options
cut to save space --// Project1.dpr
Target PostBuildEvent:
      IF EXIST "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages
\Studio16\ecc32.exe" "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages
\Studio16\ecc32.exe"       "--el_alter_exe=C:\Projects\Project1.dproj;.\Debug\Win32
\Project1.exe" "--el_config=Project1_Custom_Config.eof"
    EurekaLog Command-Line Compiler v7.0.1.0 for Delphi 15.0
    ----------------------------------------------------------------

    Loading EurekaLog options...
    EurekaLog postprocessor start...
    EurekaLog's code was added
    EurekaLog's options were added
    EurekaLog's data was added
      File size before:     2'159'616
      File size after:      2'185'216
      File size diff:       +25'600
      Debug info size:      287'554
      Symbols size:         58
      Functions size:       4
      Stripped size:        -138'240
      Number of units:      209
      Number of procedures: 10'136
      Number of lines:      28'124
      Total time:           00:00:00.639
        Compilation time:   00:00:00.026
        Prepare time:       00:00:00.015
        Post-process time:  00:00:00.597
        Events time:        00:00:00.001
      Memory usage:
        Allocated:          7'576'806
        RAM:                29'999'104
        Private:            27'066'368
        Virtual:            105'299'968
    EurekaLog postprocessor end
Build succeeded.
    0 Warning(s)
    0 Error(s)
Time Elapsed 00:00:01.91
```

Step 3: [optional] configuring alternative profiles

Finally, you may create any number of additional configuration profiles. The first thing you need to do - is to decide if you want EurekaLog for this configuration or not. The difference is that you need different project options set for different cases. As well as different unit set.

First, conditional directives. Go to "Project" / "Options" and look to "Delphi Compiler" / "Conditional Defines" option. Now, if you want EurekaLog for this profile - add EUREKALOG conditional symbol. If you don't want EurekaLog for this profile - remove EUREKALOG conditional symbol. Repeat this step for each profile of your project that you're going to use.

Second, the options of the project. EurekaLog requires certain options to be set in order to work. Also, some option may increase or decrease detalization of EurekaLog. So, if you want to use EurekaLog in certain profile - then you have to setup all required options manually. Please, read this article 58 to know what options must be set. For other profiles (in which EurekaLog will not be used) you can set options as you desire, there are no limitations.

Third, included units - those will be handled by `--el_prepare` switch which is used above.

Fourth, you have to create .eof file for **each** configuration profile and save it with corresponding name.

At last - make sure that pre/post-build events that we set in options at previous step are applied for all profiles (you can do this by entering command to Base profile and checking that it wasn't overwritten by another profiles).

Notes:

- You can use EBase unit to test whenever EurekaLog was enabled for your application or not. This unit is specially designed to be included in any application without including full EurekaLog's code.

Now, do a test - switch to different configuration profiles, make a build, run application and test it.

Note: it's recommended to make a full rebuild when changing profiles or target platform.

Conclusion

This article explained the basics of manual control over configuration of your project.

As a side note - let's discuss relation between settings and configuration profile names:

- In classic application: the debug profile has maximum debug options set, and the release profile has minimum debug options set.
- It's different for applications with exceptions tracers 40. Exception tracer requires more info than you usually use for debugging sessions (think about "Use Debug DCUs" option). So typically it's reversed now: you want maximum options for release version of your application and medium (moderate) options for debugging. Therefore, you can either swap profiles (i.e. use "debug" profile for the release version of your application) or to completely re-setup options between profiles.

See also:

- Storing EurekaLog options 440

- ecc32/emake command line options 432
- Using external configuration 443
- Compiling your project with and without EurekaLog 445
- Read-only projects 449
- Working with configurations 439

11.4 Using EurekaLog in DLL

These articles will describe pitfalls and "gotcha"s for using and handling exceptions in DLLs. It will also discuss using exception tracer tools in DLLs.

1. Introduction 455
2. What is the proper way to handle exceptions in DLLs 457
3. Using exception tracer tool in DLLs 473
4. Using exception tracer with frameworks in DLLs 484
5. Using exception tracer with COM objects 488
6. Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
7. Using EurekaLog with non-Embarcadero DLLs (Microsoft Visual Studio, etc.) 496
8. Enabling debug information for system DLLs 504
9. Configuring project for leaks detection 508

11.4.1 Introduction

1. What is DLL 455
2. What is exception 456
3. How DLLs report about failures 457

See also:

- What is the proper way to handle exceptions in DLL 457
- Creating bug reports for DLL exceptions 470

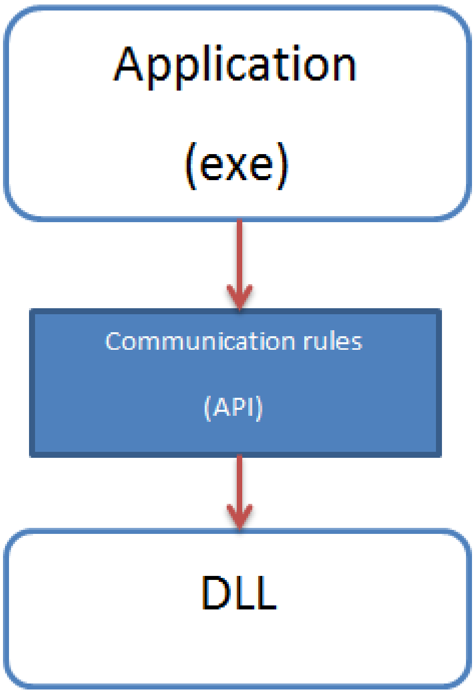
11.4.1.1 What is DLL

DLL is a library of functions. DLL can not be run as normal application - because it lacks entry point. It is not a solid program, it is a collection of functions. It requires someone else to call functions.

Application (.exe file) can load DLL (.dll file) and call a function from it. DLL is isolated file, not related to application. Therefore, DLL and application can be written in different programming languages.

Application is often called a "host application" or just "host" . Both DLL and host are also called "executable modules" or just "modules". Sometimes "application" is referred to host with loaded DLLs (not just the host itself).

DLL and application need to understand each other. So, a set of rules must be established to communicate. Such set of rules is called "API".



API is a contract between host and DLL

API can be developed by you or some other developers/company. If you're the API developer - then you can decide how DLL will work with host. If you're not the API developer - then you can only follow already established rules, but you can't invent your own rules.

See also:
- Using frameworks as wrapper for API 458
- Using system or 3rd party API 459
- Developing your own API 467

11.4.1.2 What is exception

Exception is represented by an object (class instance) in most modern high-level programming languages. This means that exceptions can be inherited from base classes, as well as be extended with arbitrary properties.

Since exception is a way to interrupt normal execution path of a code - it requires support from hardware level. Modern CPUs provides such support. However, user-mode applications do not have direct access to the hardware. Therefore, operating system provides method to use exceptions on particular hardware. This is called SEH ("Structured Exception Handling") in Windows.

Exception on operating system level is represented by its address, code, options ("flags") and up to 15 4-byte integers ("params"). High-level programming languages use SEH and this low-level representation as basis for their own exception handling. For example, exception in high-level programming language (i.e. exception object) is implemented as OS exception with special code (for example: `$EEECFADE` for Delphi) and pointer to object is stored in exception params. Exceptions with other codes are wrapped in generic class (`EExternalException` for Delphi).

Short conclusion:
1. There are 3 levels of exceptions support: hardware, OS, and programming language.
2. User-mode code has access to OS and language levels.
3. OS exceptions are compatible among all programming languages.
4. Language exceptions are specific to programming language and could not be properly used in another programming language.

See also:
- What is the proper way to handle exceptions in DLL 457

11.4.1.3 How DLLs report about failures

Remember that object and class implementations are specific for programming language and compiler. I.e. a Delphi application doesn't know how to work with objects/classes from (for example) Microsoft C++ (and visa versa). This means that hi-level exception raised in DLL could not be properly handled by host application, unless both DLL and host are compiled by the same compiler and exception class uses the virtual destructor.

Also note that mixing OS and language exceptions within same module is confusing/ problematic thing.

Therefore, APIs for DLLs usually do not use exceptions as a way to report errors. Instead: functions can use error codes - such as numeric codes, success/failure flags (booleans) and so on. There are de facto standard ways to report errors - provided by operating system (for example: `GetLastResult`, `HRESULT` - on Windows). However, 3rd party DLLs may use arbitrary error reporting method.

See also:
- What is the proper way to handle exceptions in DLL 457

11.4.2 What is the proper way to handle exceptions in DLL

As you should already understood by now 455: rule #1 when working with exceptions in DLLs is "never let exception escape DLL". That because caller may not know how to work with exception object generated by different programming language. For example, a Delphi .exe file have no idea to to read exception message from Microsoft C++ exception; it doesn't know how to properly release exception object after exception is handled. Therefore, all exceptions in DLL functions must be captured and handled by translating them to error code or other error signature as required by DLL API.

How this should be done? That highly depends on what your DLL API is. This also depends on what framework you do use. There are 3 possible cases:
1. You develop DLL by using a framework 458. For example: you write a control panel applet by using VCL. Or you write ISAPI module by using IntraWeb.
2. You develop DLL for already established API without using a ready framework 459. For example: you write a plugin for 3rd party application (like Total Commander). Or you write a global system hook (which requires DLL).
3. You develop both DLL and API specification 467. For example: you write your own DLL to be used by different applications.

See also: Creating bug reports for DLL exceptions 470.

What if I don't want to follow best practice rules?

You may want **not** to follow this rule ("never let exception escape DLL"). For example, you are sure that both host and all DLLs are always compiled by the same compiler version. Such usage case usually means using packages instead of DLLs. However, you may want to use DLLs for some other reasons.

In this case you can instruct EurekaLog to handle exceptions from other modules. You can enable this behavior by disabling "Capture stack only for exceptions from current module" option 237. You should probably disable chained exceptions support 573 for DLLs that let exceptions escape DLL and be handled by the caller. This feature requires ability to track life time of exceptions objects. This is not possible for general case (e.g. host and DLL are compiled by different compilers and there is no assist from RTL for tracking exception objects). This feature *may* work in some specific configurations. See this article 48 for more information.

Such usage case means using a single instance of exception tracer in application 474. Host

.exe must have exception tracer with above mentioned options changed and enabled support for all necessary debug information formats 355. DLLs must have debug information source, but no exception tracer.

DLLs can:

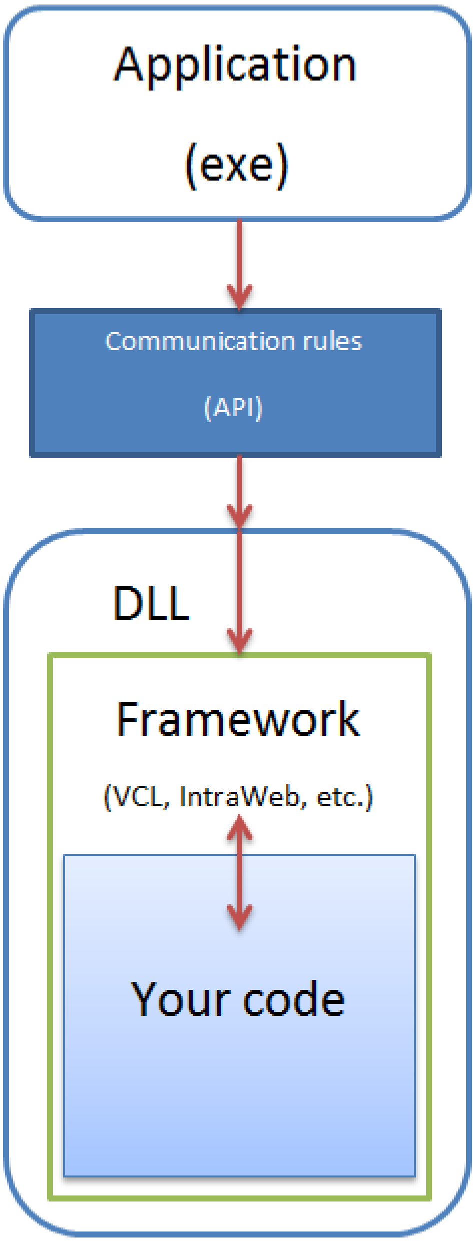
- be post-processed by EurekaLog 410 with "DLL" profile 368;
- be post-processed by JCL 412 (without having JclHookExcept active);
- be post-processed by madExcept 413 (without exception tracer activation);
- supply .map 410/.tds 411 files (this is only useful for IDEs without any exception tracer tool installed);
- supply PDB/DBG files 412;
- Non-Embarcadero DLL can be post-processed by EurekaLog based on output from 3rd party compiler 496;

See also:

- Creating bug reports for DLL exceptions 470
- Configuring call stack 48
- List of supported debug information formats 409

11.4.2.1 Framework

This is the simplest case - because all pitfalls are already handled by a framework. All your code is called by the framework. All exception from your code are handled by the framework. Framework handles exceptions and convert them to something (what is required by the API).



Framework takes care of passing exceptions between host and DLL

In this case you can just write your code as you usually do. Framework will provide a default handling and error reporting. Some frameworks also allow you to alter default handling (useful for customizations). You should refer to the documentation of your framework if you want to do such customizations. Usually, there is some sort of global `Application.OnException` event, which you may assign to your handler's code.

Note: some frameworks handles exceptions within DLL by showing error message in DLL and passing "fail" to the caller. Some frameworks leaves decision about what do to with the exception to the caller - which may show error message for exception or may do something else (like re-try or perform alternative solution). Therefore, the creation of bug reports for exceptions from DLL is not always an easy question. Some possible approaches are illustrated in this article 470.

11.4.2.2 System or 3rd party API

This case is a more complex. Basically, you need to study the API and figure out how you should report about errors in your function. You can not use arbitrary nor default way -

because API is already established by someone. It's not you who develop API. You only develop a DLL.

Note: when you write EurekaLog-enabled DLL to be used in non EurekaLog-enabled host (such as Explorer, Internet Explorer, Microsoft Office, etc.) - you have to compile your DLL with "Standalone DLL" profile 369. See also 480.

Let's consider a little example. Suppose that you want to write a global system hook - the one that is installed via `SetWindowsHookEx` function. Global hook requires you to place your handler code inside DLL, so that DLL can be injected in all running programs (which makes the hook a global one).

Naturally, API (i.e. communication rules between OS and your code) is already established - it's defined by Microsoft (as a developer of hooking functions). Therefore, the first thing that you should do is to study documentation for the functions. You pass a pointer to your handler's code via second argument in `SetWindowsHookEx` function (`lpfn` param). Prototype of the handler depends on what kind of hook do you want to use. Let's use `WH_GETMESSAGE` hook for this example. This means that we must study description of GetMsgProc callback.

The important part for error handling looks like this:

> If code is less than zero, the hook procedure must return the value returned by CallNextHookEx.
>
> If code is greater than or equal to zero, it is highly recommended that you call CallNextHookEx and return the value it returns; otherwise, other applications that have installed WH_GETMESSAGE hooks will not receive hook notifications and may behave incorrectly as a result. If the hook procedure does not call CallNextHookEx, the return value should be zero.

In other words, your code could not report any failure reason. All that you can do is either return 0 or return whatever `CallNextHookEx` returns.

Therefore, your DLL code must looks at least like this:

```
library Project2;

uses
  Windows;

function MyHook(Code: Integer; _wParam: WPARAM;
  _lParam: LPARAM): LRESULT; stdcall;
begin
  try
    if Code >= 0 then
    begin
      // <- your code
    end;
  except
    // There is no way to report errors,
    // so we must handle all exceptions
  end;
  Result := CallNextHookEx(Code, _wParam, _lParam);
end;
```

Usually it's not a good idea to silently hide all exceptions. If API doesn't allow you to report about errors - then you should at least implement some kind of logging, so you can store information about exception in the log. Some possible solutions for except block are examined in this article 470.

Let's see another example. Suppose you're writing a control panel applet without using any framework. This means that you must write and register DLL. DLL must export CPlApplet function. This function will be used for all communication between OS and your code. Description of CPlApplet says:

> The return value depends on the message.
> For more information, see the descriptions of the individual Control Panel messages.

This means that you also must study each message from the system that you want to process. Luckily, most messages require you to handle errors in the same way:

> If the CPlApplet function processes this message successfully, the return value is zero; otherwise, it is nonzero.

So, you should write your DLL at least like this:

```
library Project2;

uses
  Windows;

function CPlApplet(hwndCPl: HWND; uMsg: UINT;
  lParam1, lParam2: LPARAM): LongInt; stdcall;
begin
  try
    case uMsg of
      ...
    end;

    Result := 0;
  except
    Result := 1;
  end;
end;

exports
  CPlApplet;

end.
```

Since you can't report what is actual report is - a good idea would be to report error to the user. We can safely do this because control panel applet is a single interactive GUI application. Showing error as dialog box is not a good idea for non-interactive applications (such as services) or code that may be used multiple times (such as global hook).

```
library Project2;

uses
  Windows;

function CPlApplet(hwndCPl: HWND; uMsg: UINT;
  lParam1, lParam2: LPARAM): LongInt; stdcall;
begin
  try
    case uMsg of
      ...
    end;

    Result := 0;
  except
    on E: Exception do
    begin
      MessageBox(hwndCPl, PChar(E.Message),
        'Error', MB_OK or MB_ICONERROR);
      Result := 1;
    end;
  end;
end;

exports
  CPlApplet;

end.
```

Of course, since you can show error message - you can show the entire bug report instead:

```
library Project2;

uses
  // EurekaLog units for "Standalone DLL":
  EMemLeaks,
  EResLeaks,
  EDialogWinAPIMSClassic,
  EDialogWinAPIEurekaLogDetailed,
  EDialogWinAPIStepsToReproduce,
  EDebugExports,
  ExceptionLog7,

  // EurekaLog units for our code
  EExceptionManager,

  Windows;

function CPlApplet(hwndCPl: HWND; uMsg: UINT;
  lParam1, lParam2: LPARAM): LongInt; stdcall;
begin
  try
    case uMsg of
      ...
    end;

    Result := 0;
  except
    on E: Exception do
    begin
      EExceptionManager.Handle(E, ExceptAddr);
      Result := 1;
    end;
  end;
end;

exports
  CPlApplet;

end.
```

(Of course, your DLL must be compiled with EurekaLog enabled, DLL must use "Standalone DLL" profile 369, you should configure dialogs, bug reports, sending).

Please note that code above is just example. Not all messages to control panel applet have the same requirements. You should study description of each message that you're going to handle in your code. For example, `CPL_INIT` message has different requirements:

> If initialization succeeds, the CPlApplet function should return nonzero. Otherwise, it should return zero.
> If CPlApplet returns zero, the controlling application ends communication and releases the DLL containing the Control Panel application.

Therefore, you need to use such code to handle `CPL_INIT` message:

```
library Project2;

uses
  Windows;

function CPlApplet(hwndCPl: HWND; uMsg: UINT;
  lParam1, lParam2: LPARAM): LongInt; stdcall;
var
  SuccessCode, FailureCode: LongInt;
begin
  // "If initialization succeeds, the CPlApplet function should return nonzero.
  // Otherwise, it should return zero."
  if uMsg = CPL_INIT then
  begin
    SuccessCode := 1;
    FailureCode := 0;
  end
  else
  // "If the CPlApplet function processes this message successfully,
  // the return value is zero; otherwise, it is nonzero."
  begin
    SuccessCode := 0;
    FailureCode := 1;
  end;

  try
    case uMsg of
      ...
    end;

    Result := SuccessCode;
  except
    on E: Exception do
    begin
      MessageBox(hwndCPl, PChar(E.Message),
        'Error', MB_OK or MB_ICONERROR);
      Result := FailureCode;
    end;
  end;
end;

exports
  CPlApplet;

end.
```

Next example would be a Shell extension. Shell extensions are implemented as COM objects. That means that you need to write and register a DLL, which follows COM rules. A COM rule for error handling is to use HRESULT as return value of any method. There are two ways to work with HRESULT. First one is quite direct: you write a function/method that returns HRESULT and you convert each exception to HRESULT value:

```
...

function ConvertExceptionToHRESULT(const E: Exception): HRESULT;
begin
  Result := E_FAIL; // <- this is just a simple example
  // See HandleSafeCallException function from ComObj unit
  // to see more complicated example
end;

type
  ICopyHook = interface(IUnknown)
  ['{000214FC-0000-0000-C000-000000000046}']
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PWideChar; dwSrcAttribs: DWORD;
      pszDestFile: PWideChar; dwDestAttribs: DWORD): HRESULT; stdcall;
  end;

  TMyHook = class(TInterfacedObject, ICopyHook)
  protected
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PWideChar; dwSrcAttribs: DWORD;
      pszDestFile: PWideChar; dwDestAttribs: DWORD): HRESULT; stdcall;
  end;

function TMyHook.CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PWideChar; dwSrcAttribs: DWORD; pszDestFile: PWideChar;
      dwDestAttribs: DWORD): HRESULT; stdcall;
begin
  try
    // your code

    Result := S_OK;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

function DllCanUnloadNow: HRESULT; stdcall;
begin
  try
    if { it's OK to unload DLL } then
      Result := S_OK
    else
      Result := S_FALSE;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

...
```

The second way is to use Delphi wrapper for HRESULT. Delphi compiler provides assisting for HRESULT retu

```
function Funcensten1(... some arguments ...): HRESULT; stdcall;
function Funcensten2(... some arguments ...; out AResult: TSomeType): HRESULT; stdc
```

has the same protype and the same calling convention as such function:

```
procedure Funcensten1(... some arguments ...); safecall;
function Funcensten2(... some arguments ...): TSomeType; safecall;
```

In other words, the above code fragments are binary compatible with each other. So, for example, DLL may use first code block and host may use second code block - and both will work correctly.

The difference between HRESULT/stdcall and safecall headers is assisting from Delphi compiler. Each safecall function and method automatically handles all exceptions within itself. Moreover, each call to safecall function/method automatically converts HRESULT return value back to exception.

So, the second way to work with HRESULT is:

```
...

type
  ICopyHook = interface(IUnknown)
  ['{000214FC-0000-0000-C000-000000000046}']
    procedure CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PWideChar; dwSrcAttribs: DWORD; pszDestFile: PWideChar;
      dwDestAttribs: DWORD); safecall;
  end;

  TMyHook = class(TInterfacedObject, ICopyHook)
  protected
    procedure CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PWideChar; dwSrcAttribs: DWORD; pszDestFile: PWideChar;
      dwDestAttribs: DWORD); safecall;
  end;

procedure TMyHook.CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
  pszSrcFile: PWideChar; dwSrcAttribs: DWORD; pszDestFile: PWideChar;
  dwDestAttribs: DWORD); safecall;
begin
  // your code
end;

function DllCanUnloadNow: HRESULT; stdcall;
// Unfortunately, it's not possible to customize return code
// to be S_FALSE for simple function.
// Otherwise DllCanUnloadNow could have been written like this:
//  procedure DllCanUnloadNow; safecall;
begin
  try
    if { it's OK to unload DLL } then
      Result := S_OK
    else
      Result := S_FALSE;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

...
```

Converting exception to HRESULT value will be done automatically by Delphi's RTL code.

Notes:

- A more detailed description of using COM can be found in this article 488;
- Barebone converting to HRESULT in ConvertExceptionToHRESULT may be insufficient for your needs. It's possible to customize it by overriding SafeCallException method. See Delphi help for more information;
- Possible implementations of ConvertExceptionToHRESULT with creation of bug reports are discussed in this article 470;
- COM also allow you to ship additional information with exception. See SetErrorInfo function.

See also:

- Using exception tracer with COM objects 488
- Creating bug reports for DLL exceptions 470

11.4.2.3 Your own API

When you want to develop a new DLL which will be used by many applications ("common DLL") or if you want to write an application which may be extended with 3rd party DLLs ("plugins") - then you need to develop API, i.e. set of rules which will be used to communications between host and DLLs.

COM - a default solution to design API

It's a good idea to provide an informative and easy way to report and handle errors. An easy solution would be to use COM 488. That's because COM is relatively modern API, which provides a decent way to work with errors. COM also has support in many frameworks.

Second best bet - HRESULT via interfaces or functions

If you think that COM is an "overkill" for your application, then you have to develop your own API. It would be a good idea to use HRESULT as base of error handling part in your API. That's because HRESULT offers a good range of possible error values, it has additional support in Delphi (via safecall) and it's familiar for many Windows developers.

So, functions from your DLL may looks like this:

```
library Project2;

uses
  Windows;

procedure Init; safecall;
// the same as:
// function Init: HRESULT; stdcall;
begin
  // your code
end;

function DoSomething(A1: Integer; const A2: WideString): Integer; safecall;
// the same as:
// function DoSomething(A1: Integer; const A2: WideString;
//                      out AResult: Integer): HRESULT; stdcall;
begin
  // your code
  Result := { ... };
end;

procedure Done; safecall;
// the same as:
// function Done: HRESULT; stdcall;
begin
  // your code
end;

exports
  Init, DoSomething, Done;

end;
```

As an alternative to a "safecall compiler magic" - you may write the same code as this:

```
library Project2;

uses
  Windows;

function Init: HRESULT; stdcall;
// the same as:
// procedure Init; safecall;
begin
  try
    // your code

    Result := S_OK;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

function DoSomething(A1: Integer; const A2: WideString;
                     out AResult: Integer): HRESULT; stdcall;
// the same as:
// function DoSomething(A1: Integer; const A2: WideString): Integer; safecall;
begin
  try
    // your code
    AResult := { ... };

    Result := S_OK;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

function Done: HRESULT; stdcall;
// the same as:
// procedure Done; safecall;
begin
  try
    // your code

    Result := S_OK;
  except
    on E: Exception do
      Result := ConvertExceptionToHRESULT(E);
  end;
end;

exports
  Init, DoSomething, Done;

end;
```

Both implementations are binary compatible with each other and do the same thing. The difference is that second implementation allows you to control exception handling.

Note: it's also a good idea to use interfaces instead of simple functions in your DLLs. Interfaces allow you to customize safecall handling by overriding SafeCallException method. Interfaces also allow you to simplify memory management and avoid using shared memory manager.

Possible ways to handle exceptions within DLLs

ConvertExceptionToHRESULT is some function that you need to write which will handle exceptions and converts failure reasons to HRESULT codes. The simplest implementation may look like this:

```
function ConvertExceptionToHRESULT(E: Exception): HRESULT;
begin
  Result := E_FAIL;
end;
```

(The same function is used as default action for TObject.SafeCallException method.)

Obviously, such primitive function will ignore any exception's info and just report "something went wrong" to the caller. A more complex implementation of this function can be found in System.Win.ComObj unit - see HandleSafeCallException function. (This function is used as default action for TComObject.SafeCallException method.)

```
uses
  ComObj;

function ConvertExceptionToHRESULT(E: Exception): HRESULT;
const
begin
  Result := HandleSafeCallException(E, ExceptAddr, GUID_NULL, GUID_NULL, '');
end;
```

This example will additionally pass error message and help context via IErrorInfo interface. This example pass empty values for COM-related information (error ID, source ID and help file name). You can use these values as you want in your own API. COM applications should fill these values with actual information (this task is done automatically for VCL-based COM objects - see TComObject.SafeCallException method).

Creating bug reports for DLL exceptions

Please note that no implementation of ConvertExceptionToHRESULT in the above examples creates bug report for the exception.

Creating bug reports for DLL exceptions is not an easy question - because final handling of the exception is not under your control. It is the caller of your DLL who decides what to do with the exceptions from your DLLs. Surely, exception from your DLL may be handled as usual: by showing error message to end user, asking to send bug report to developers, etc. However, exception from your DLL may also be silently handled by the caller and failed action will be repeated. Or the caller may try to execute fallback method with alternative solution (for example, the code that tries to set application for auto-launch may write to HKEY_LOCAL_MACHINE registry key. If this action will fail due to application being run under limited user account - the code may switch to HKEY_CURRENT_USER key, e.g. re-try action with other params).

For these reasons you can not simply show error dialog and ask to send bug report - because:

- If exception from your DLL will be handled as error by the caller - then the error message will appear twice: first from your DLL as bug report, second - from the caller;
- If exception from your DLL will be handled as non-error by the caller (i.e. the caller may re-try action or try alternative solutions) - then your DLL will show "false" error message (user will see error dialog - even though requested action will be completed by the caller).

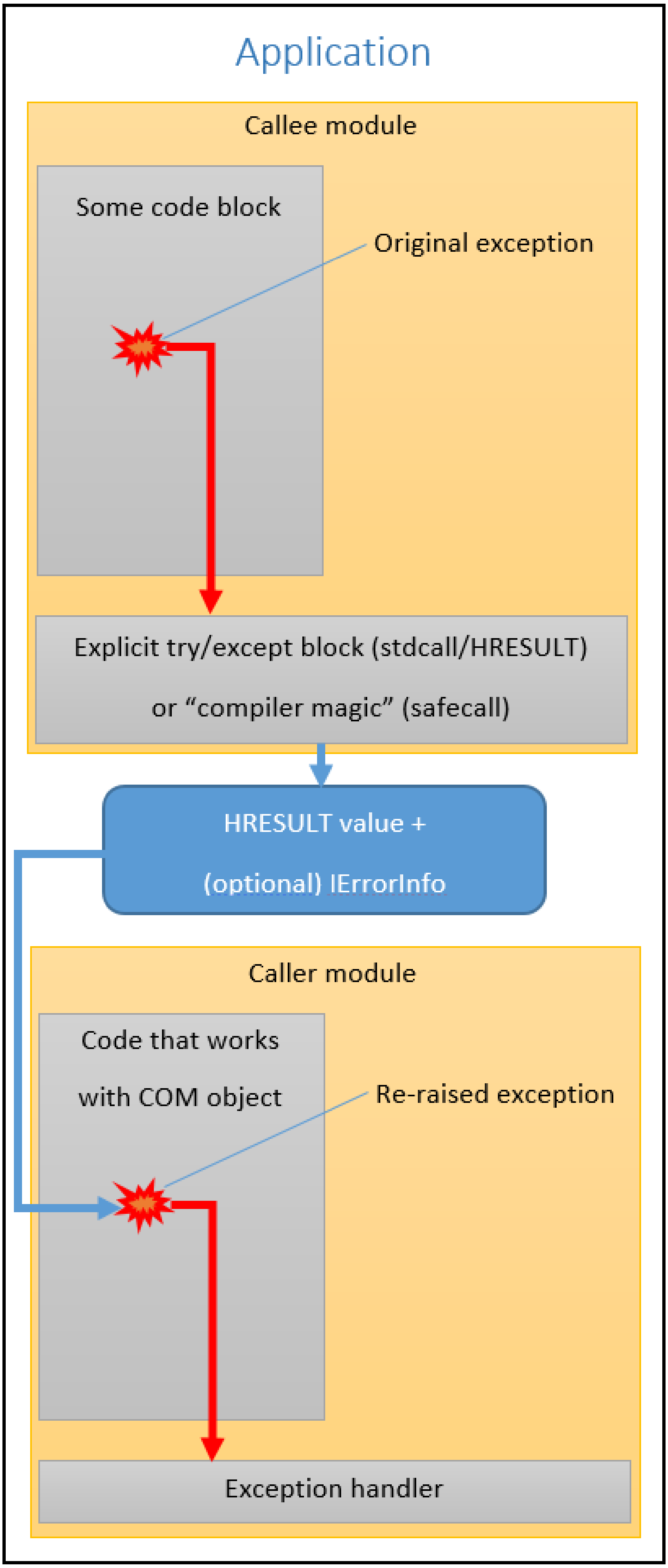
Some possible solutions to this problem are explored in this article 470.

See also:

- Using exception tracer with COM objects 488
- Creating bug reports for DLL exceptions 470

11.4.2.4 Creating bug reports for DLL exceptions

Exceptions should never leave DLLs 457. This means that exceptions should be handled in exported functions and converted to some error sign (a flag, an integer error code, HRESULT code, etc.). The entire life-time of any exception from DLL can be illustrated by this example:


Application
Callee module
Some code block
Original exception
Explicit try/except block (stdcall/HRESULT)
or "compiler magic" (safecall)
HRESULT value +
(optional) IErrorInfo
Caller module
Code that works
with COM object
Re-raised exception
Exception handler

Exception's life-time in COM applications

This example uses COM application for illustration, but the similar is true for any other good designed DLL with minor adjustments.

Notice that such approach will not create bug reports for the exceptions. Each exception from DLL (callee) is converted to "error code". No error message is shown to user. Error message shown to user comes from second exception - the one that was raised by caller/ host.

Creating bug reports for DLL exceptions is not an easy question - because final handling of the exception is not under your control. It is the caller of your DLL who decides what to do with the exceptions from your DLLs. Surely, exception from your DLL may be handled as usual: by showing error message to end user, asking to send bug report to developers, etc. However, exception from your DLL may also be silently handled by the caller and failed action will be repeated. Or the caller may try to execute fallback method with alternative solution (for example, the code that tries to set application for auto-launch may write to HKEY_LOCAL_MACHINE registry key. If this action will fail due to application being run under limited user account - the code may switch to HKEY_CURRENT_USER key, e.g. re-try action with other params).

For these reasons you can not simply show error dialog and ask to send bug report - because:

- If exception from your DLL will be handled as error by the caller - then the error message will appear twice: first from your DLL as bug report, second - from the caller;
- If exception from your DLL will be handled as non-error by the caller (i.e. the caller may re-try action or try alternative solutions) - then your DLL will show "false" error message (user will see error dialog - even though requested action will be completed by the caller).

You can try to use different approaches for creating bug reports for exceptions in DLLs. A very simple approach would be to append call stack to exception message:

```
uses
  ComObj;

function ConvertExceptionToHRESULT(E: Exception): HRESULT;
const
begin
  Result := HandleSafeCallException(E + E.StackTrace, ExceptAddr, GUID_NULL, GUID_N
end;
```

(Works for both "DLL" and "Standalone DLL" profiles.)

This is a very simple, but still a good way. A great advantage is that this approach will work in any host - even if host have no exception tracer enabled. If exception from your DLL will be recognized as an error by the caller, then the caller will show error message - and your error message includes call stack. This will give you information about the problem. And this will work for any host - including Internet Explorer (ActiveX), Microsoft Office (COM plugins), etc.

However, the limitation of the above method is that it will not provide you a full bug report (with environment information, CPU/Assembly dump, etc.). It will only show a call stack.

Other possible solution would be to configure DLL to create bug report files silently - i.e. without dialogs and sending reports and to create new bug report for each exception that leaves DLL:

```
uses
  ComObj, EAppType;

function ConvertExceptionToHRESULT(E: Exception): HRESULT;
const
begin
  _ExceptionManagerHandle(E, ExceptAddr);
  Result := HandleSafeCallException(E, ExceptAddr, GUID_NULL, GUID_NULL, '');
end;
```

(Works for both "DLL" and "Standalone DLL" profiles.)

Therefore, all bug reports from DLL will be collected in bug reports output folder for DLL:

- If your .exe host is not EurekaLog-enabled - you can't automatically show and/or send DLL bug reports. However, you may manually open bug reports to study bugs.
- If your .exe host is EurekaLog-enabled - it can attach these files to its own bug report as additional files:
  - o [Recommended] If your application uses a single instance of exception tracer 474 (i.e. host .exe has EurekaLog, DLL does not have EurekaLog - all DLLs are compiled with "DLL" profile 368). This means that both host and DLLs will use the same exception tracer and the same options. Thus, all bug reports will be stored in the same file. You need to allow multiple bug reports in bug report file 264 by setting "Max. reports in one file" option to value greater than 1. You should also set "Send entire bug report file with multiple reports" option 304 to send all reports together.
  - o [Not recommended] If your application uses multiple instances of exception tracer 480 (i.e. both host .exe and DLLs have EurekaLog - all DLLs are compiled with "Standalone DLL" profile 369). You can supply a name to bug report file as "help file name":

```
function ConvertExceptionToHRESULT(E: Exception): HRESULT;
const
begin
  _ExceptionManagerHandle(E, ExceptAddr);
  Result := HandleSafeCallException(E, ExceptAddr, GUID_NULL, GUID_NULL,
    ExceptionManager.Info(E, ExceptAddr).Options.OutputLogFile(''));
end;
```

    A caller may use help file name property to figure out file name with bug report from DLL.

You can design and implement any other solution to create bug report files. You can combine the above solutions.

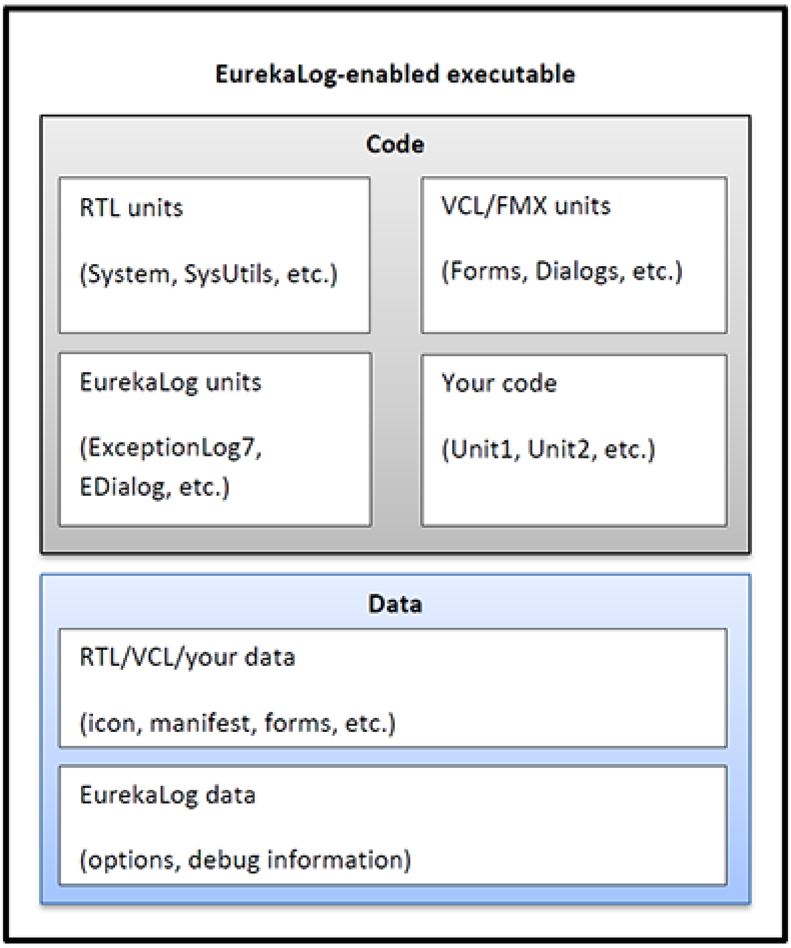
See also:

- Using exception tracer with COM objects 488
- Using exception tracer with system/3rd party API 459
- Developing DLL API 467

11.4.3 Using exception tracer tool in DLLs

Many developers prefer to use exception tracer tool in their DLLs. Exception tracer collects information about problems in your code, allowing you to diagnose failures more easily.

Remember what exception tracer does to your application 38:



EurekaLog-enabled executable

Exception tracer includes its code in your module. It also injects some data - debug information and options. Both (code and data) are required for exception tracer to function.

When you have more than just single executable module - things become interesting. Exception tracer could be inserted into one module or into each module:
1. There is single instance of exception tracer in application 474
2. Each module has its own exception tracer code 480

First case is good when you can afford enabling exception tracer in host application. Centralized management will allow you to reduce performance cost when you have many DLLs. For example, consider application with 50 DLLs (keep in mind "plugins" scenario). Each exception must be analyzed by exception tracer. If each DLL has its own exception tracer - then each exception will be analyzed 50 times. A good idea would be to have only one instance of exception tracer, so information is collected only once. Any DLL can ask central exception tracer for information about exception.

Second case is good when host application is out of your control. Since you can not use exception tracer in host - then the only choice left is to add it to DLL. Each DLL will have its own isolated exception tracer. Examples are ActiveX (host will be Internet Explorer), COM (host can be Microsoft Office), etc.

See also:
- Using exception tracer with COM objects 488

11.4.3.1 Single instance of exception tracer

This case require you to enable exception tracer for host application. You should do this in the same way as you do it for typical application without any DLLs. For example, if you have

VCL Forms application as the host - then you need to enable EurekaLog for host application and set application type to "VCL Forms Application". This will add EurekaLog code and data into final .exe file. It would also set hook for `Forms.TApplication.HandleException` method, which will allow to automatically handle exceptions without writing any code.

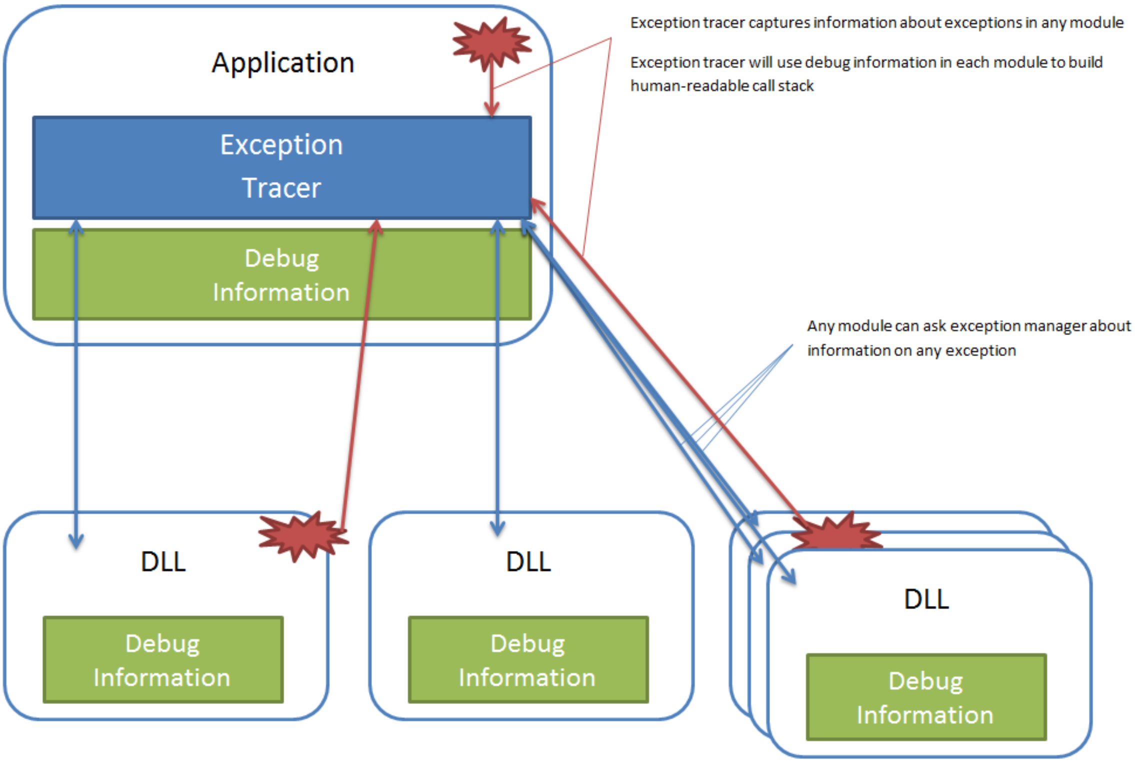
Now, the host has exception tracer. The host must be configured with different settings depending on your API design:

- If your API follows the best practice 457 ("never let exception escape DLL") - you don't need to alter any other options;
- If your API does not follow the best practice 457 ("never let exception escape DLL") - you have to disable "Capture stack only for exceptions from current module" 237 and chained exceptions support 573 options as explained here 457.

Each DLL must also has EurekaLog enabled and application type must be set to "**DLL**". Such settings will inject debug information into DLL, but will not include exception tracer code. Rather DLL will ask host application for information. Please note that majority of EurekaLog settings will be ignored for DLL, since there will be no EurekaLog code in your DLL.

Note: it's not strictly necessary to enable EurekaLog for DLLs in this example. You can just supply debug information 409 and keep EurekaLog off. For example, DLLs can:

- be post-processed by EurekaLog 410 with "DLL" profile 368;
- be post-processed by JCL 412 (without having JclHookExcept active);
- be post-processed by madExcept 413 (without exception tracer activation);
- supply .map 410/.tds 411 files (this is only useful for IDEs without any exception tracer tool installed);
- supply PDB/DBG files 412;
- Non-Embarcadero DLL can be post-processed by EurekaLog based on output from 3rd party compiler 496;


Application
Exception Tracer
Debug Information
Exception tracer captures information about exceptions in any module
Exception tracer will use debug information in each module to build human-readable call stack
Any module can ask exception manager about information on any exception
DLL
Debug Information
DLL
Debug Information
DLL
Debug Information

Host application loads multiple DLLs with "DLL" profile

Let's see this on practice. Create a new VCL application and place buttons to invoke functions from DLL.

```
...

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    FDLL: HMODULE;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  FDLL := LoadLibrary('Project2.dll');
  Win32Check(FDLL <> 0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  FreeLibrary(FDLL);
  FDLL := 0;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise Exception.Create('Error Message');
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  P: procedure;
begin
  P := GetProcAddress(FDLL, 'Test1');
  Win32Check(Assigned(P));
  P;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  P: procedure;
begin
  P := GetProcAddress(FDLL, 'Test2');
  Win32Check(Assigned(P));
  P;
end;
```

```
...
```

Code is pretty simple: we load DLL on form's creating and unload it when form is gone. There are 3 buttons on the form to invoke different testing routines. First button raises exception in application itself. The 2nd and 3rd buttons raise exceptions in DLL.

Don't forget to enable EurekaLog for host and set application type to "VCL Forms Application". That's all.

Now, create a new DLL project:

```
library Project2;

uses
  // Added automatically
  EAppDLL, // "DLL" profile

  // Added manually
  Windows,
  SysUtils,
  Classes;

{$R *.res}

procedure Test1;
begin
  // This is just example! It's highly not recommended to do this.
  raise Exception.Create('Error Message');
end;

procedure Test2;
var
  Msg: String;
begin
  // Recommended way: handle exceptions, do not let them escape DLL
  try
    raise Exception.Create('Error Message');
  except
    on E: Exception do
    begin
      Msg := E.ToString + sLineBreak + E.StackTrace;
      MessageBox(0, PChar(Msg), 'Error', MB_OK or MB_ICONSTOP);
    end;
  end;
end;

exports
  Test1,
  Test2;

end.
```

This simple DLL exports 2 functions to test exceptions. First function raises exception and lets it escape DLL, so it will be catched by caller. In our test example caller would be the host application. Such approach is not recommended - as it's already explained above: you should never let exceptions escape DLL. This is done only for example (illustration). It will work correctly for our case, because DLL and exe are both compiled by the same compiler. This will not work properly for generic case. So, it's only suitable for testing.

Note: if you want to raise exception in DLL and catch it by the caller - then do not use DLLs. Use packages instead. Using packages will guarantee compatible compiler for host and DLL (package). It's also generally less problematic with all cross-modules communications.

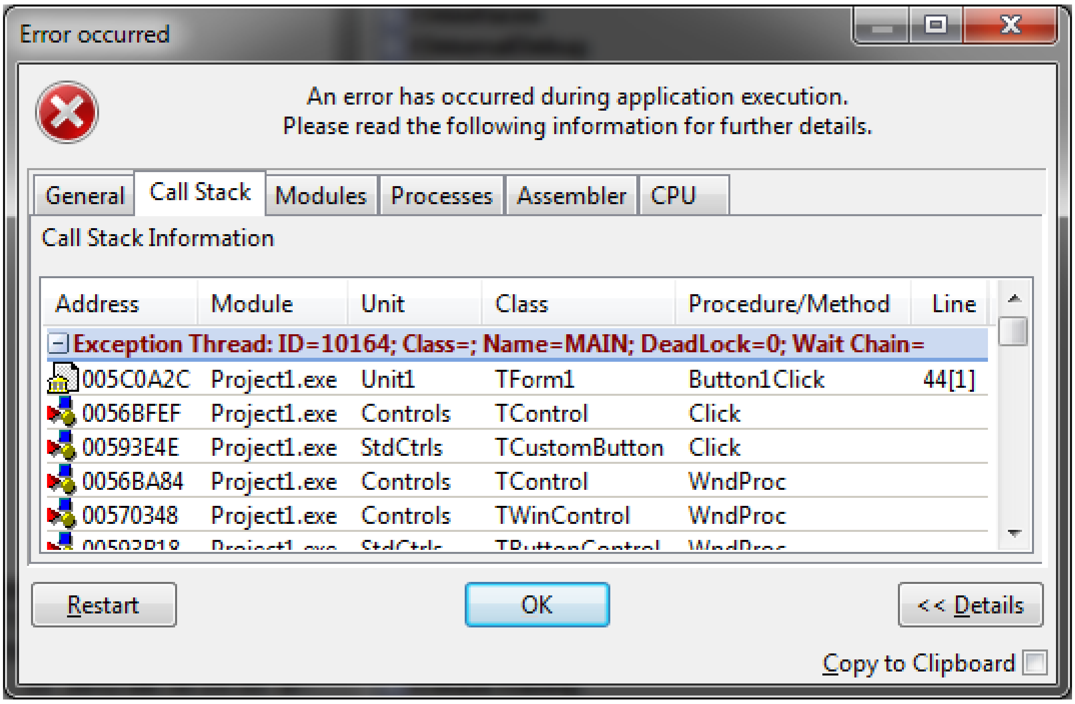
Second function illustrate more correct approach: we catch exceptions in function and handle them somehow. For this example we will do very simple handling: just display error message with stack trace. More proper approach was discussed above: you should use some kind of error indication (such as boolean flag, HRESULT, etc.) to indicate failure condition to the caller.

Now, enable EurekaLog for this DLL and set application type to "DLL".

Note: EAppDLL unit will be added automatically when you set profile to "DLL". This unit contains default callbacks into host to ask for exception's info and handling. You may use your own custom callbacks instead.

Compile both host and DLL project, run host application.

Hit buttons 1-3.

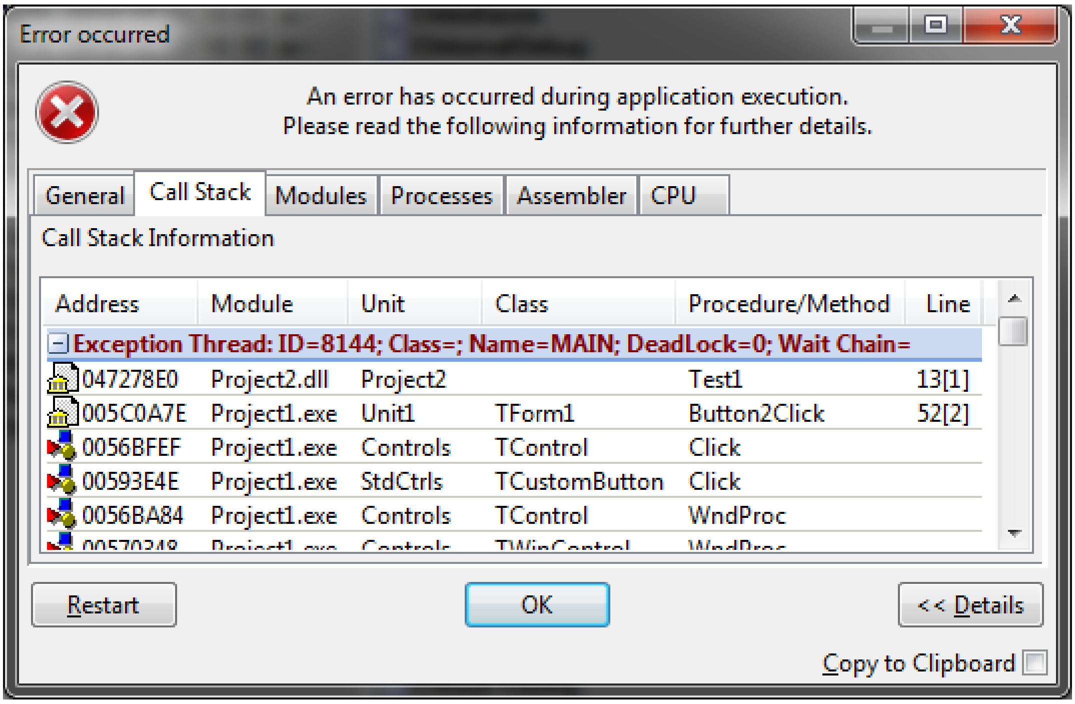


An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Exception Thread: ID=10164; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
005C0A2C	Project1.exe	Unit1	TForm1	Button1Click	44[1]
0056BFEF	Project1.exe	Controls	TControl	Click	
00593E4E	Project1.exe	StdCtrls	TCustomButton	Click	
0056BA84	Project1.exe	Controls	TControl	WndProc	
00570348	Project1.exe	Controls	TWinControl	WndProc	
00593B18	Project1.exe	StdCtrls	TButtonControl	WndProc	

Typical exception in host application
Call stack shows only items for exe

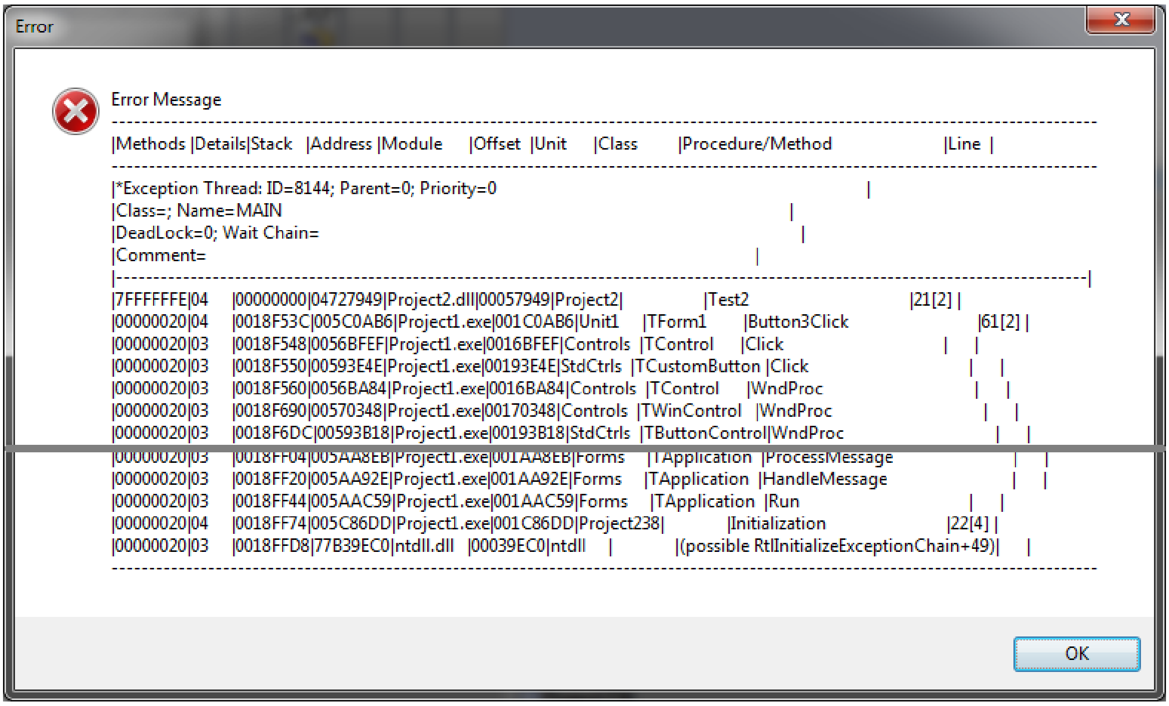


An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=8144; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
047278E0	Project2.dll	Project2		Test1	13[1]
005C0A7E	Project1.exe	Unit1	TForm1	Button2Click	52[2]
0056BFEF	Project1.exe	Controls	TControl	Click	
00593E4E	Project1.exe	StdCtrls	TCustomButton	Click	
0056BA84	Project1.exe	Controls	TControl	WndProc	
00570348	Project1.exe	Controls	TWinControl	WndProc	

Exception escaped DLL
Call stack shows mixed exe/DLL lines
Notice line numbers for routine in DLL



Error Message

```
-----------------------------------------------------------------------------------------------------
|Methods |Details|Stack  |Address |Module     |Offset  |Unit      |Class        |Procedure/Method                  |Line |
-----------------------------------------------------------------------------------------------------
|*Exception Thread: ID=8144; Parent=0; Priority=0                                             |
|Class=; Name=MAIN                                                                    |
|DeadLock=0; Wait Chain=                                                                |
|Comment=                                                                          |
|----------------------------------------------------------------------------------------------------|
|7FFFFFFE|04    |00000000|04727949|Project2.dll|00057949|Project2|              |Test2                      |21[2] |
|00000020|04    |0018F53C|005C0AB6|Project1.exe|001C0AB6|Unit1    |TForm1       |Button3Click                  |61[2] |
|00000020|03    |0018F548|0056BFEF|Project1.exe|0016BFEF|Controls |TControl     |Click                      |     |
|00000020|03    |0018F550|00593E4E|Project1.exe|00193E4E|StdCtrls |TCustomButton |Click                      |     |
|00000020|03    |0018F560|0056BA84|Project1.exe|0016BA84|Controls |TControl     |WndProc                    |     |
|00000020|03    |0018F690|00570348|Project1.exe|00170348|Controls |TWinControl  |WndProc                    |     |
|00000020|03    |0018F6DC|00593B18|Project1.exe|00193B18|StdCtrls |TButtonControl|WndProc                   |     |
|00000020|03    |0018FF04|005AA8EB|Project1.exe|001AA8EB|Forms    |TApplication |ProcessMessage                |     |
|00000020|03    |0018FF20|005AA92E|Project1.exe|001AA92E|Forms    |TApplication |HandleMessage                 |     |
|00000020|03    |0018FF44|005AAC59|Project1.exe|001AAC59|Forms    |TApplication |Run                        |     |
|00000020|04    |0018FF74|005C86DD|Project1.exe|001C86DD|Project238|            |Initialization                |22[4] |
|00000020|03    |0018FFD8|77B39EC0|ntdll.dll   |00039EC0|ntdll    |             |(possible RtlInitializeExceptionChain+49)|     |
-----------------------------------------------------------------------------------------------------
```

Exception did not escape DLL, it was handled by DLL by displaying error message (screenshot was cut to save space)
Call stack shows mixed exe/DLL lines
Notice line numbers for routine in DLL

Please note that last case was a simple example of trivial exception handling in DLL. You may be not satisfied with looks of error dialog. You may want not just "error message", but complete bug report. To do this - you need to replace the call to `MessageBox` with a call to exception manager. Normally, it would be `ExceptionManager.Handle`. However, there is no exception manager in our DLL. We'll show how to do this in the "Working with frameworks and exception tracers in DLLs 484" article or "Creating bug reports for DLL exceptions 470" article).

See also:

- Creating bug reports for DLL exceptions 470

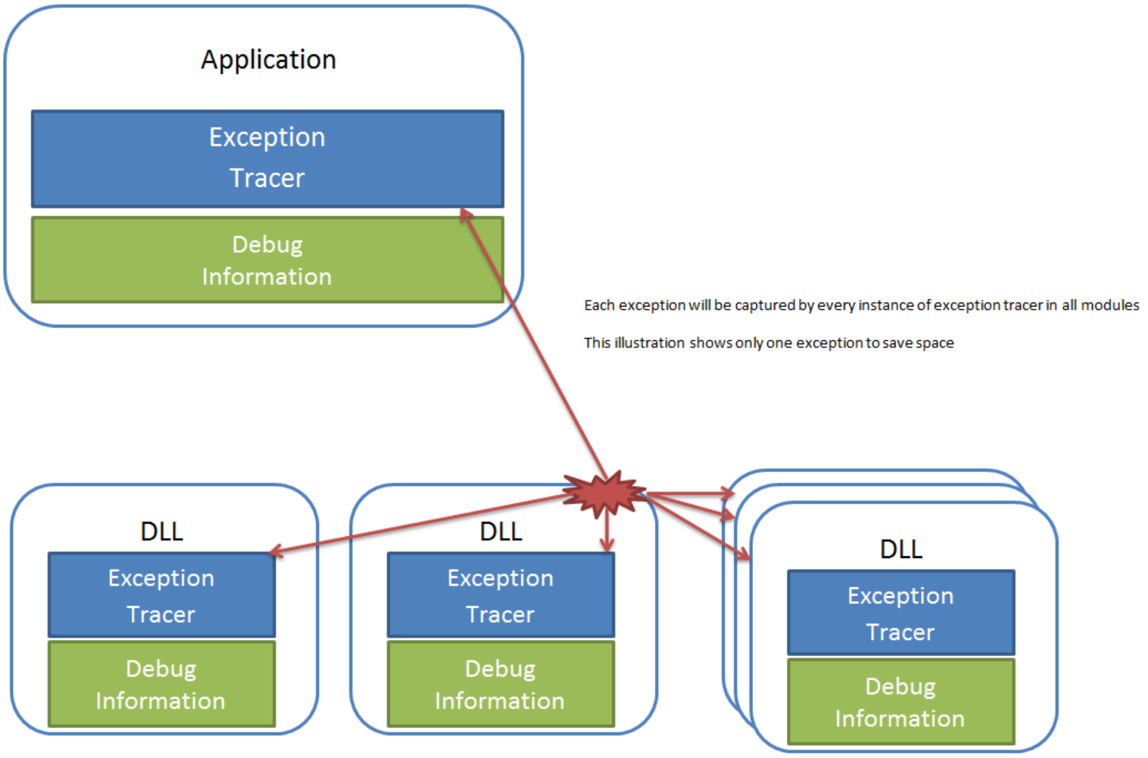
11.4.3.2 Multiple instances of exception tracer

This case does not require you to enable exception tracer for host application. You can do it, but it's not required. Typically this approach should be used only when you develop DLLs to be used in non-EurekaLog enabled host. If you have EurekaLog enable for the host - please try to implement case 1 approach 474.

Since host application do not necessary have exception tracer - you must to include tracer in each of your DLLs. Each DLL will have exception tracer. All tracers and DLLs will be independent of each other. Each exception will be catched by each exception tracer in each DLL.

Therefore, each DLL must has EurekaLog enabled and application type must be set to "**Standalone DLL**". Such settings will add exception tracer in each DLL and inject debug information.

Important notice: Windows 2000 does not provide any way to set exception hook in documented way. This means that any exception tracer have to install low-level injecting hook for internal routines. Only single module can install such hook reliably for the same routine. If two or more different modules attempt to install such hook - it will either fail or crash. Therefore, it's highly not recommended to use multiple instances of exception tracers on Windows 2000. Windows XP and above do not have such issues, because newer systems allow you to install arbitrary amount of exception hooks via documented API. This API is called VEH: Vectored Exception Handling. If you can't use single instance of exception tracer and have to support Windows 2000: we suggest to use Delphi 2009 or above and disable "Use low-level hooks" option 259. Delphi 2009 introduced better integration between application and exception tracer. It allow you to react on hi-level exceptions without need to install low-level hook (however, low-level hook still may be installed to capture CPU state). Combination of Windows 2000 and any IDE before 2009 (such as Delphi 7) will not work reliably for multiple instances of exception manager - regardless of options in those modules.



Host application loads multiple DLLs with "Standalone DLL" profile

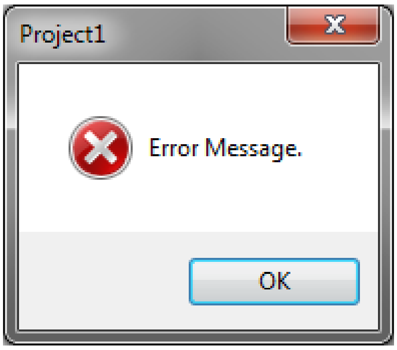
Let's see this on practice. We'll use the same host application for this example. Of course, it has EurekaLog enabled, but remember that it's not necessary. You may turn EurekaLog off for host application, if you want. Actually, let's do this for the sake of better illustration. So, open your host application project, disable EurekaLog for it and recompile (all source code will remain the same as above).

We'll use the same DLL project for this example. We'll make only few changes. Open DLL project and change application type from "DLL" to "Standalone DLL". This will also replace EAppDLL unit in uses clause with multiple EurekaLog units. Also, go to dialogs options and change "None" to any dialog that you like. We will use "MS Classic" for this example.

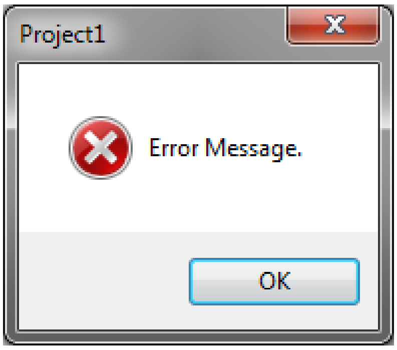
This could be all, but since now we have full exception tracer on board - why not ask it to handle exceptions? We can replace our old MessageBox with a call to exception manager. So full changed code will looks like:

```
library Project2;

uses
  // Automatically generated:
  EMemLeaks,
  EResLeaks,
  EDialogWinAPIMSClassic,
  EDialogWinAPIEurekaLogDetailed,
  EDialogWinAPIStepsToReproduce,
  EDebugExports,
  EDebugJCL,
  ExceptionLog7,

  // Added manually:
  EExceptionManager,
  Windows,
  SysUtils,
  Classes;

{$R *.res}

procedure Test1;
begin
  raise Exception.Create('Error Message');
end;

procedure Test2;
begin
  try
    raise Exception.Create('Error Message');
  except
    ExceptionManager.ShowLastExceptionData;
  end;
end;

exports
  Test1, Test2;

end.
```

Save all and recompile. Run application and hit all 3 buttons:

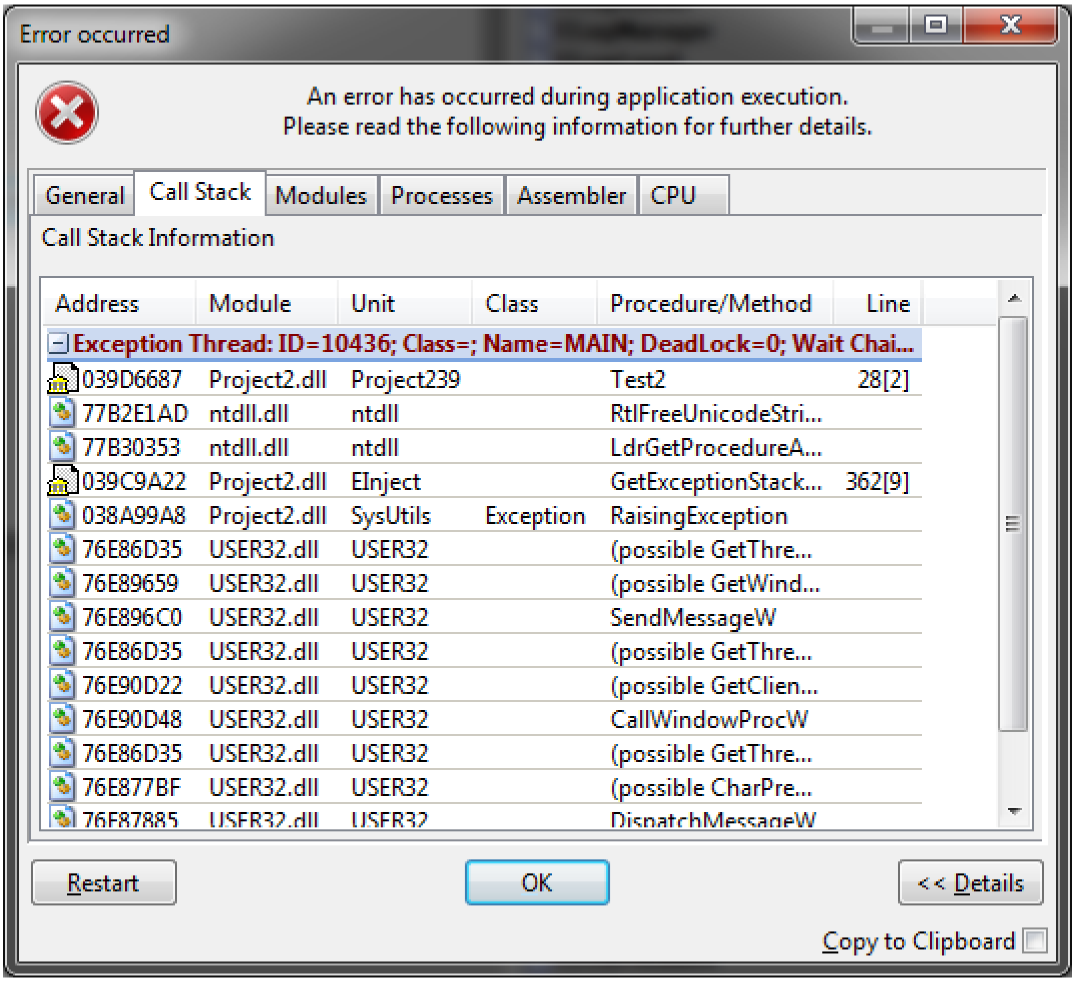


Typical exception in host application
There is no bug report, since host application do not have exception tracer



Error Message.

Exception escaped DLL
There is no bug report, since exception was caught by host application (without exception tracer)



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟Exception Thread: ID=10436; Class=; Name=MAIN; DeadLock=0; Wait Chai...					
039D6687	Project2.dll	Project239		Test2	28[2]
77B2E1AD	ntdll.dll	ntdll		RtlFreeUnicodeStri...	
77B30353	ntdll.dll	ntdll		LdrGetProcedureA...	
039C9A22	Project2.dll	EInject		GetExceptionStack...	362[9]
038A99A8	Project2.dll	SysUtils	Exception	RaisingException	
76E86D35	USER32.dll	USER32		(possible GetThre...	
76E89659	USER32.dll	USER32		(possible GetWind...	
76E896C0	USER32.dll	USER32		SendMessageW	
76E86D35	USER32.dll	USER32		(possible GetThre...	
76E90D22	USER32.dll	USER32		(possible GetClien...	
76E90D48	USER32.dll	USER32		CallWindowProcW	
76E86D35	USER32.dll	USER32		(possible GetThre...	
76E877BF	USER32.dll	USER32		(possible CharPre...	
76E87885	USER32.dll	USER32		DispatchMessageW	

Exception did not escape DLL, it was handled by DLL by displaying complete bug report
Call stack shows only lines within DLL
There is no information about exe, because exe do not have any debug infomation

This example gives you full EurekaLog support within DLL, but host .exe completely lacks any support. It doesn't even have debug information, so even exception tracer from DLL is unable to display call stack for exe. Of course, this can be fixed by enabling EurekaLog for exe. Just remember that host application is not always under your control.

See also:

- Using exception tracer with COM objects 488

11.4.4 Using exception tracer with frameworks in DLLs

The previous sections 473 assumed that you write DLLs without using any frameworks. If you use framework (such as VCL or IntraWeb), then your actions will be slightly different. That's because framework already contain some sort of exception handling code.

A general concept would be the same. You can use either "DLL" or "Standalone DLL" profiles for your DLLs. So the previous facts 473 would be the same. Additionally, you have to configure DLLs for your framework. EurekaLog has support for common cases out of the box. For example, if you use `Forms` unit in your DLL (i.e. your DLL has forms) - then you need to hook `Application.HandleException` method. This can be done by enabling "VCL Forms application" option on Advanced/Code/Hooks page in EurekaLog project options 352. This is true for both "DLL" and "Standalone DLL" profiles.

Note: "Standalone DLL" profile with "VCL Forms application" option on Hooks page is equal to "VCL Forms Application" profile. When you enable such options - DLL profile will be switched to "VCL Forms Application" profile. This is normal behavior. After all, a profile is just set of predefined options. If you change options to match another profile - it will be shown as used. There is no build-in profile for "DLL" profile with "VCL Forms application" option, so "DLL" profile will not be changed after enabling option.

Now, let's change our example to illustrate this on practice. As usual, host application will remain unchanged. All changes will be done for DLL.

1. Single instance of exception tracer 484
2. Multiple instances of exception tracer 487

See also:
- Using exception tracer with COM objects 488

11.4.4.1 Single instance of exception tracer

Create new DLL project, enable EurekaLog and set application type to "DLL". Since we're going to use forms in our DLL - go do Advanced/Code/Hooks page in EurekaLog project options 352 and enable "VCL Forms application" option.

Now, create a new form for DLL, place a button to raise exception:

```
...

type
  TForm2 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
  raise Exception.Create('Error Message');
end;

...
```

And change DLL code as:

```
library Project2;

uses
  // Automatically generated by EurekaLog
  EAppDLL,  // "DLL" profile
  EAppVCL,  // "VCL Forms application" hook

  // Added manually
  EAppType,
  Windows,
  SysUtils,
  Classes,
  Unit2 in 'Unit2.pas' {Form2};

{$R *.res}

procedure Test1;
begin
  try
    raise Exception.Create('Error Message');
  except
    on E: Exception do
      // Ask exception manager in host application to process this exception
      _ExceptionManagerHandle(E, ExceptAddr);
  end;
end;

procedure Test2;
var
  Form: TForm2;
begin
  try
    Form := TForm2.Create(nil);
    try
      Form.ShowModal;
    finally
      FreeAndNil(Form);
    end;
  except
    on E: Exception do
      // Ask exception manager in host application to process this exception
      _ExceptionManagerHandle(E, ExceptAddr);
  end;
end;

exports
  Test1, Test2;

end.
```

Normally, if you want to ask EurekaLog to process exception (display error dialog with bug report, send it to developer, etc.) - then you have to call `ExceptionManager.Handle`. However, we can not do this in our case, because we've used "DLL" profile, which means no exception tracer (and no exception manager) in our DLL. That's why we use `_ExceptionManagerHandle` function instead of `ExceptionManager.Handle`.

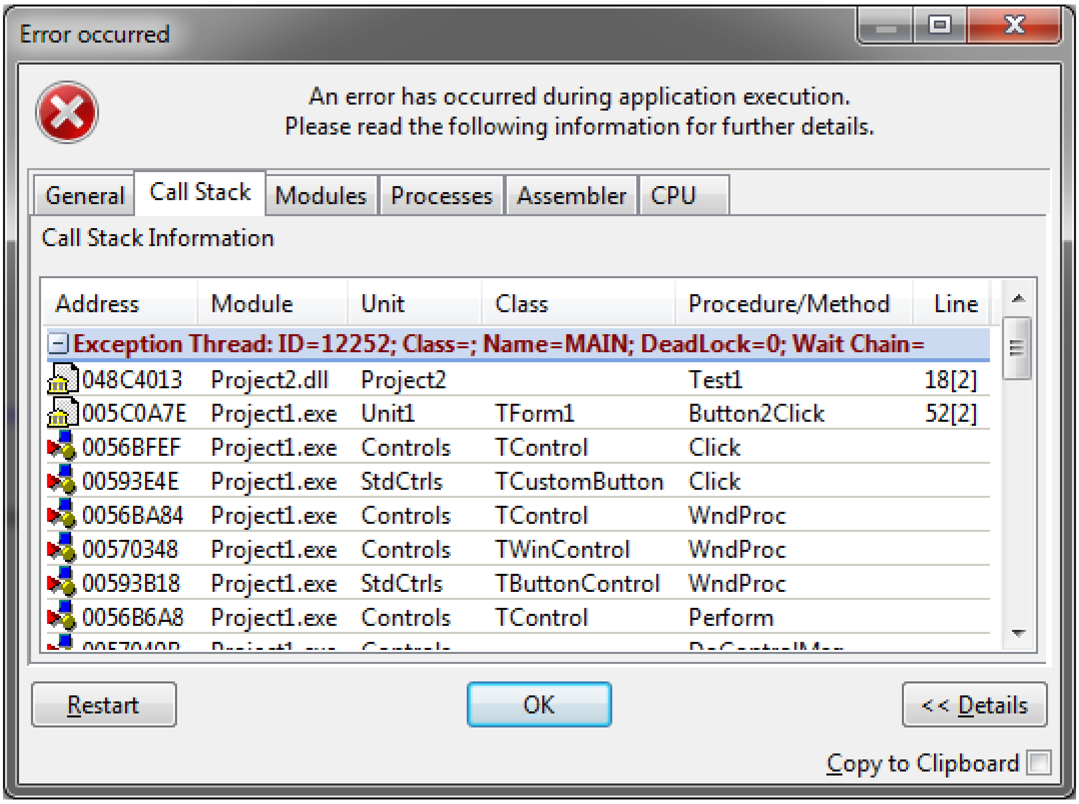
`_ExceptionManagerHandle` function is a lightweight exception manager. If there is exception tracer code in current module - the function will invoke it (i.e. `ExceptionManager.Handle`). If there is no tracer in the module - the function will try to invoke exception manager from host application. If there is no tracer in host application either - the function will act as if EurekaLog was disabled.

Therefore, you can use `_ExceptionManagerHandle` function to handle exceptions when you don't know if there will be EurekaLog in your project. This function will automatically use suitable way to process exceptions.

Okay, so the first function in our DLL will just raise exception in DLL function. The difference with first example is that we handle it properly now: there is try/except block which handles exception by asking exception manager from host application to perform full processing (displaying bug report, sending it to developer, etc.).

Second function will create and show a modal form. There is no exception inside function itself, but form contains button to raise exception. This exception will not be catched by our try/except block, because exceptions in form's event handlers are handled by VCL framework. That's why we need `EAppVCL` unit (it contains hooks for VCL). Try/except block in second function will catch exceptions only for form's creating or destroying.

That's all. Save all and compile. Run application and hit all buttons. First button is not changed at all. Second button and third button behave differently:



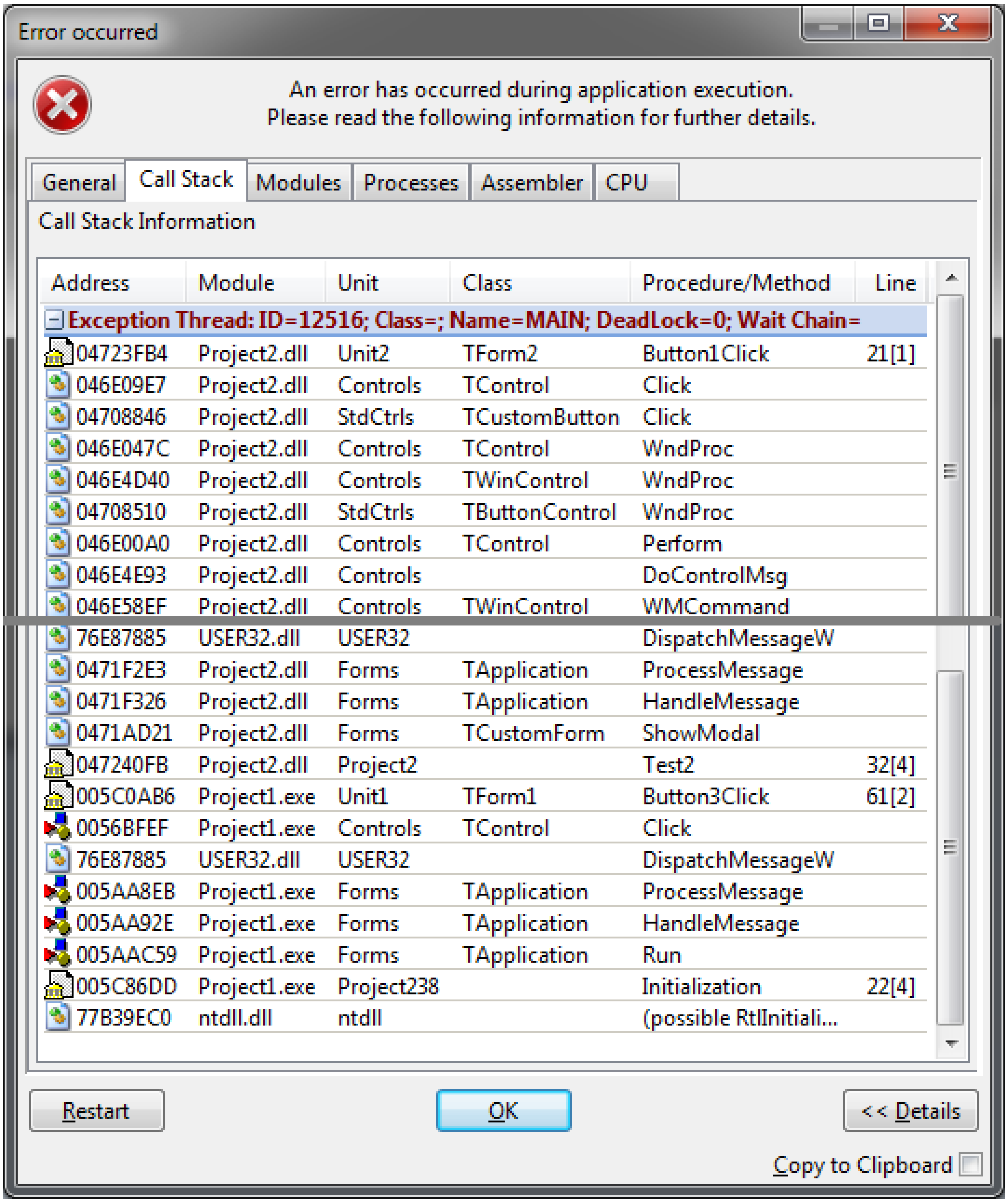
Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Exception Thread: ID=12252; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
048C4013	Project2.dll	Project2		Test1	18[2]
005C0A7E	Project1.exe	Unit1	TForm1	Button2Click	52[2]
0056BFEF	Project1.exe	Controls	TControl	Click	
00593E4E	Project1.exe	StdCtrls	TCustomButton	Click	
0056BA84	Project1.exe	Controls	TControl	WndProc	
00570348	Project1.exe	Controls	TWinControl	WndProc	
00593B18	Project1.exe	StdCtrls	TButtonControl	WndProc	
0056B6A8	Project1.exe	Controls	TControl	Perform	

Button #2: Exception did not escape DLL, it was handled by DLL by displaying complete bug report
Call stack shows mixed exe/DLL lines



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟Exception Thread: ID=12516; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
04723FB4	Project2.dll	Unit2	TForm2	Button1Click	21[1]
046E09E7	Project2.dll	Controls	TControl	Click	
04708846	Project2.dll	StdCtrls	TCustomButton	Click	
046E047C	Project2.dll	Controls	TControl	WndProc	
046E4D40	Project2.dll	Controls	TWinControl	WndProc	
04708510	Project2.dll	StdCtrls	TButtonControl	WndProc	
046E00A0	Project2.dll	Controls	TControl	Perform	
046E4E93	Project2.dll	Controls		DoControlMsg	
046E58EF	Project2.dll	Controls	TWinControl	WMCommand	
76E87885	USER32.dll	USER32		DispatchMessageW	
0471F2E3	Project2.dll	Forms	TApplication	ProcessMessage	
0471F326	Project2.dll	Forms	TApplication	HandleMessage	
0471AD21	Project2.dll	Forms	TCustomForm	ShowModal	
047240FB	Project2.dll	Project2		Test2	32[4]
005C0AB6	Project1.exe	Unit1	TForm1	Button3Click	61[2]
0056BFEF	Project1.exe	Controls	TControl	Click	
76E87885	USER32.dll	USER32		DispatchMessageW	
005AA8EB	Project1.exe	Forms	TApplication	ProcessMessage	
005AA92E	Project1.exe	Forms	TApplication	HandleMessage	
005AAC59	Project1.exe	Forms	TApplication	Run	
005C86DD	Project1.exe	Project238		Initialization	22[4]
77B39EC0	ntdll.dll	ntdll		(possible RtlInitiali...	

Button #3: Exception was raised by form. It was handled by VCL.
EurekaLog hook displays full bug report
Call stack shows mixed exe/DLL lines

11.4.4.2 Multiple instances of exception tracer

Important notice: Windows 2000 does not provide any way to set exception hook in documented way. This means that any exception tracer have to install low-level injecting hook for internal routines. Only single module can install such hook reliably for the same routine. If two or more different modules attempt to install such hook - it will either fail or crash. Therefore, it's highly not recommended to use multiple instances of exception tracers on Windows 2000. Windows XP and above do not have such issues, because newer systems allow you to install arbitrary amount of exception hooks via documented API. This API is called VEH: Vectored Exception Handling. If you can't use single instance of exception tracer and have to support Windows 2000: we suggest to use Delphi 2009 or above and disable "Use low-level hooks" option 259. Delphi 2009 introduced better integration between application and exception tracer. It allow you to react on hi-level exceptions without need to install low-level hook (however, low-level hook still may be installed to capture CPU state). Combination of Windows 2000 and any IDE before 2009 (such as Delphi 7) will not work reliably for multiple instances of exception manager - regardless of options in those modules.

Open/create DLL, enable EurekaLog and set application type to "Standalone DLL". Since we're going to use forms in our DLL - go do Advanced/Code/Hooks page 352 in EurekaLog project options and enable "VCL Forms application" option. Also change dialog to "MS Classic" or any other desired type.

Note: a combination of "Standalone DLL" profile + "VCL Forms application" hook will set the same options as "VCL Forms Application" profile. That's why you'll see "VCL Forms Application" instead of "Standalone DLL" in "Application type" option when you open project settings next time. That's totally expected behavior. You can also initially only switch profile to "VCL Forms Application" and do nothing else - that's because this profile will set dialogs and turn on hooks for VCL.

The code for both DLL and exe remain unchanged from previous example 484. Run application and hit the buttons. You should see the same behavior and dialogs as in previous example.

Note that even if visual appearance seems the same - the internals are working differently now. DLL now has its own exception tracer. `_ExceptionManagerHandle` function will just invoke `ExceptionManager.Handle` in DLL. It will not try to communicate with exe host.

See also:
- Using exception tracer with COM objects 488

11.4.5 Using exception tracer with COM objects

COM objects are implemented in DLLs. Therefore, COM server projects with EurekaLog should follow usual rules for DLLs 457. Additionally, COM enforces additional restrictions on error handling: therefore a most complex issue for COM object is how to report bug reports 470. Many COM objects are multi-threaded. This means that you must follow guidelines for multi-threaded applications 547.

To summarize restrictions on error handling:
- COM objects must handle each exception in the same method, do not allow exception escape to the caller;
- Each COM object's method must be a function. Each such function must return an `HRESULT` value. Each such function must have the `stdcall` calling convention;
- COM methods must return S_OK (zero) for success calls, E_UNEXPECTED for unknown errors, or any specific error/success code for known problems.
- (Optional) COM objects may report additional details for failure error codes via `IErrorInfo` `interface`.

Additionally, COM objects are "plugins". I.e. a single application (.exe host) may operate with several COM object, each being loaded into the same process. Therefore, it's important to be sure that multiple COM objects with exception tracers will not conflict with each other. Since COM object can be used in any application host (with or without EurekaLog or exception tracer) - it means that each COM server project must be standalone. I.e. there must be exception tracer instance in COM server project. Thus, if application loads several COM objects - there will be multiple instances of exception tracer active.

Important notice: Windows 2000 does not provide any way to set exception hook in documented way. This means that any exception tracer have to install low-level injecting hook for internal routines. Only single module can install such hook reliably for the same routine. If two or more different modules attempt to install such hook - it will either fail or crash. Therefore, it's highly not recommended to use multiple instances of exception tracers on Windows 2000. Windows XP and above do not have such issues, because newer systems allow you to install arbitrary amount of exception hooks via documented API. This API is called VEH: Vectored Exception Handling. If you can't use single instance of exception tracer and have to support Windows 2000: we suggest to use Delphi 2009 or above and disable "Use low-level hooks" option 259. Delphi 2009 introduced better integration between application and exception tracer. It allow you to react on hi-level exceptions without need to install low-level hook (however, low-level hook still may be installed to capture CPU state). Combination of Windows 2000 and any IDE before 2009 (such as Delphi 7) will not work reliably for multiple instances of exception manager - regardless of options in those modules. If you can afford to not support Windows 2000 - it's recommended to enable "Use low-level hooks" option 259.

Normally, you develop COM servers in Delphi and C++ Builder by using assist from VCL. I.e. you use File / New / ActiveX library + File / New / COM Object. Resulting code will use ComServ unit with TComServer class and ComObj unit with TComObject class.

"stdcall + HRESULT" vs. "safecall"

RAD Studio offers you a helper to easily follow COM requirements outlined above. Normally, a COM method should look something like this:

```
type
  TSampleObject = class(TTypedComObject, ISampleObject)
  protected
    function DoSomething(Param1: Integer; const Param2: WideString; out Rslt: WideS
  end;

function TSampleObject.DoSomething(Param1: Integer; const Param2: WideString; out R
begin
  try
    Rslt := IntToStr(Param1 + StrToInt(Param2));

    Result := S_OK;
  except
    on EConvertError do
      Result := E_INVALIDARG
    else
      Result := E_UNEXPECTED;
  end;
end;

// ...

var
  Obj: ISampleObject;
  Value: WideString;
// ...
  OleCheck(Obj.DoSomething(1, '4', Value));
```

As you can see: you have to place explicit try/except block to handle all exceptions. You have to write error handling code by yourself for each method. You have to return HRESULT value. Function's real result must be converted to last output argument. You also have to manually check result code (HRESULT) on caller's side (notice a call to OleCheck in the example above). Obviously, this is not a very convenient way.

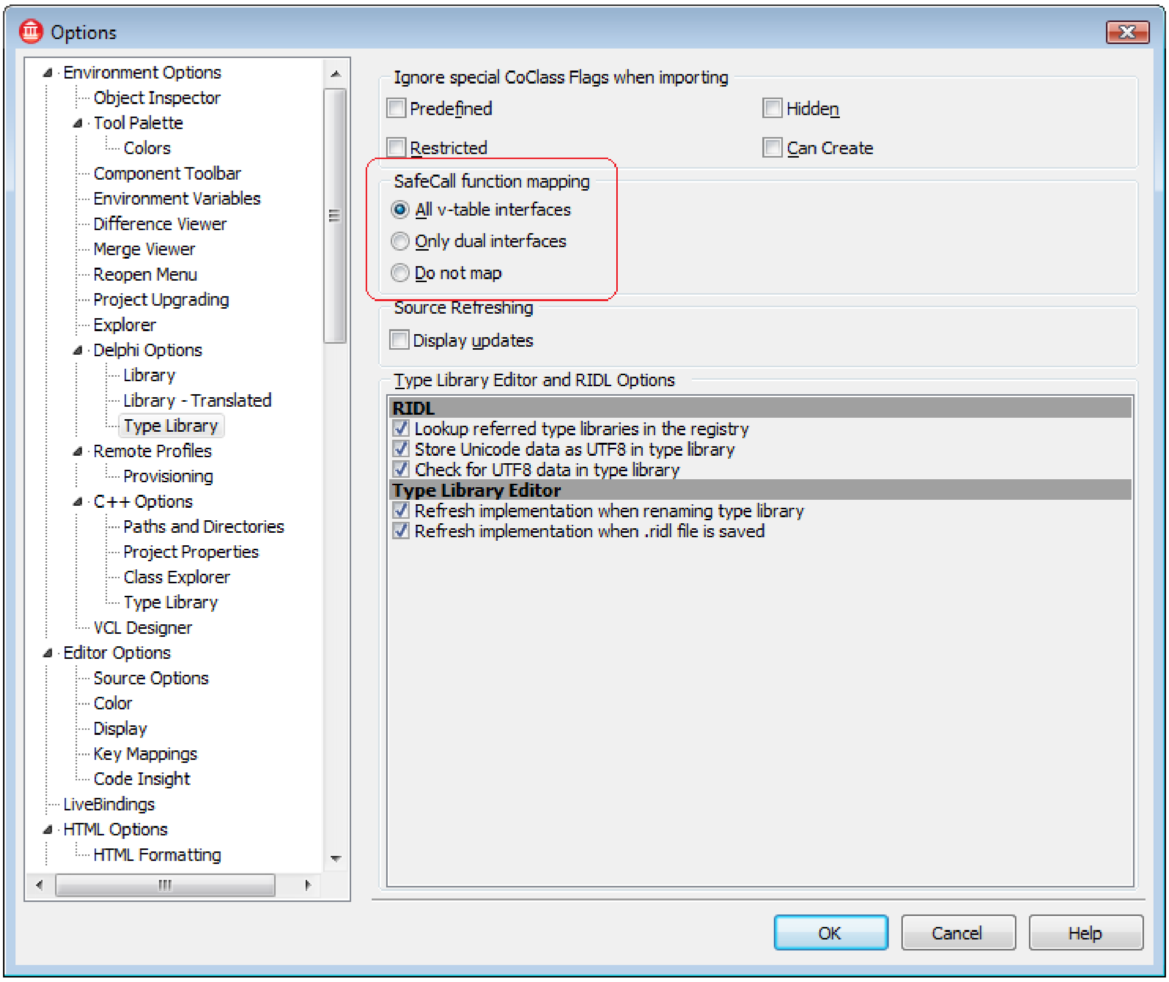
Fortunately, there is a special safecall calling convention. The safecall convention implements exception 'firewalls'. On Win32, this implements interprocess COM error notification. Therefore, you can use the following approach instead:

```
type
  TSampleObject = class(TTypedComObject, ISampleObject)
  protected
    function DoSomething(Param1: Integer; const Param2: WideString): WideString; sa
  end;

function TSampleObject.DoSomething(Param1: Integer; const Param2: WideString): Wide
begin
  Result := IntToStr(Param1 + StrToInt(Param2));
end;

// ...

var
  Obj: ISampleObject;
  Value: String;
// ...
  Value := Obj.DoSomething(1, '4');
```

The both code samples are binary compatible. I.e. both have the same method prototype. The difference is that second example uses "compiler magic". Compiler automatically inserts a hidden try/except block to catch all exceptions within method and convert them to appropriate error code. Each exception is handled by SafeCallException virtual method of current object. `Safecall` calling convention allows you to write your code in usual way: use `Result` as you would normally do, no need to place exception handling blocks, no need to check return codes on caller's side.

For more information about `safecall`, see System.SysUtils.ESafecallException, System.SafeCallErrorProc, and System.TObject.SafeCallException.

When you create new COM object (server side) or import type library (client side) - you can create/import methods either in "stdcall + HRESULT" form or in "safecall" form. The IDE's behavior is controlled by corresponding options:



Ignore special CoClass Flags when importing

□ Predefined □ Hidden

□ Restricted □ Can Create

SafeCall function mapping

● All v-table interfaces

○ Only dual interfaces

○ Do not map

Source Refreshing

□ Display updates

Type Library Editor and RIDL Options

RIDL

☑ Lookup referred type libraries in the registry

☑ Store Unicode data as UTF8 in type library

☑ Check for UTF8 data in type library

Type Library Editor

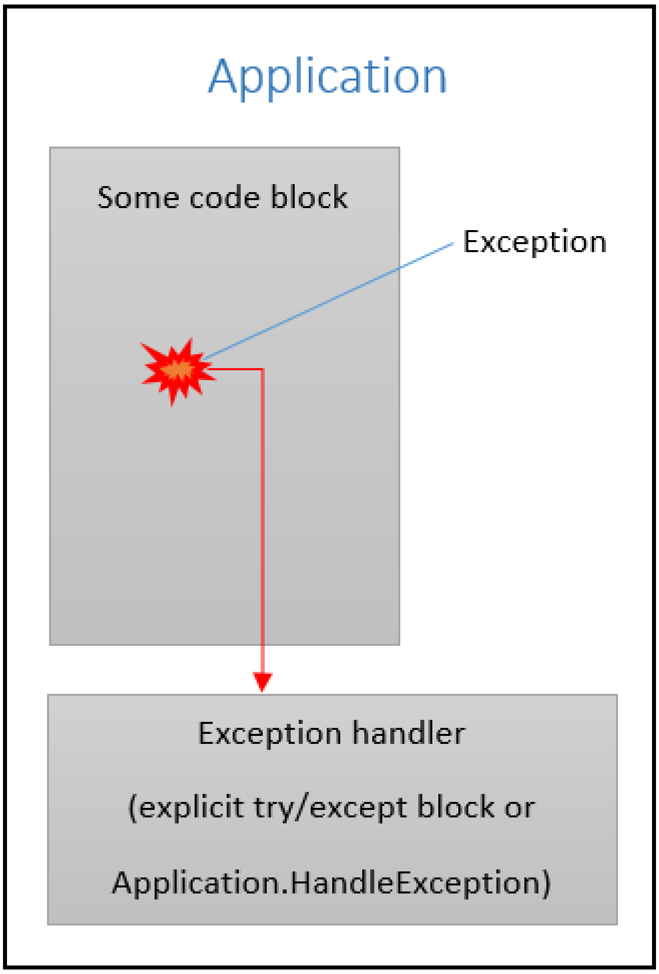
☑ Refresh implementation when renaming type library

☑ Refresh implementation when .ridl file is saved

SafeCall options under "Tools / Options" menu item

Exception's life-time with COM objects

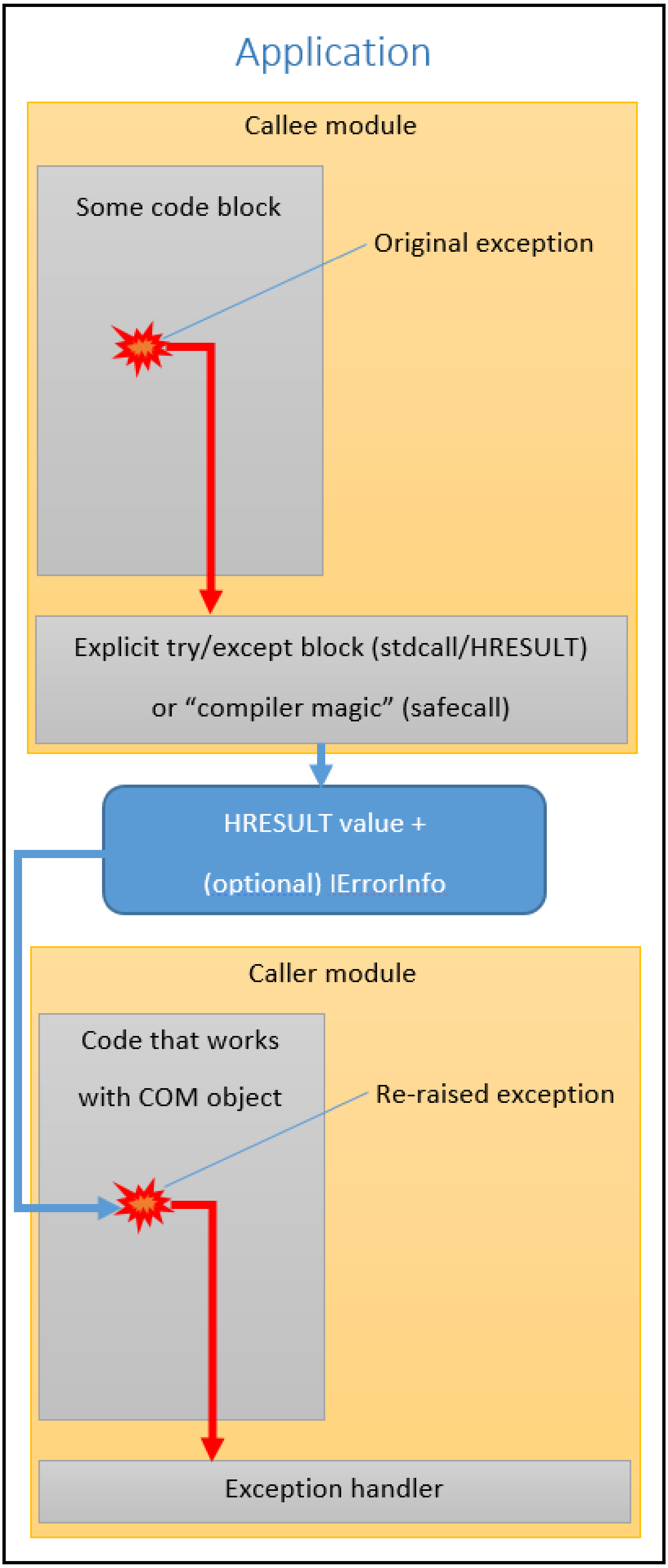
Exception's life-time is significantly different in COM applications (compared to typical applications). First, let's see an illustration for a normal VCL application:



Exception's life-time in VCL application

Here: exception is raised by some code. Exception tracer detects this moment via hooks and creates call stack. Then exception is passed through one or more exception handlers (such as `finally` and `except` blocks). Eventually, exception may be processed and deleted or (in case of unknown exceptions) it could be passed to default handler. Exception tracer intercepts default handler and create bug report for exception (shows dialog, send via Internet, etc.).

Now, let's see how this scenario will be different for COM objects:


Application
Callee module
Some code block
Original exception
Explicit try/except block (stdcall/HRESULT)
or "compiler magic" (safecall)
HRESULT value +
(optional) IErrorInfo
Caller module
Code that works
with COM object
Re-raised exception
Exception handler

Exception's life-time in COM applications

Here: "callee module" is COM server project (DLL). Caller module is .exe host (however, it can also be a DLL, package or even another COM object for generic case). Again, exception is raised by some code. However, it can not be passed directly to exception's handler of the caller. Because rule #1 for COM is: do not let exceptions escape your code. So, this exception is catched by "firewall" ("compiler magic" for `safecall` methods or explicit except block for `stdcall`/`HRESULT`).

Notice important difference: each conceptual exception is represented by two different exceptions - one for callee side and another one for caller side. This means that there will be two bug reports, two dialogs, etc. For this reason **it's recommended to disable error dialogs for COM server projects** (switch dialog to "None" 267) and **keep visual dialog for end user side** (.exe host). COM server project will create bug report file, but will not show any error dialog and will not submit error to developers. Then, exception will appear in caller module (as `HRESULT` value + `IErrorInfo` interface). Caller will create another bug report, show dialog, and send report to developer. If caller is awared about exception tracing features in failed COM object, then caller may include bug report from failed COM object as attachment to its own bug report.

Please note that exception travel chain may include more than one "exception -> HRESULT -> exception" transformation. For example, your .exe host calls method from COM object, this method calls another COM method (in the same object or in some another COM object). If the last callee raise exception, then this exception will travel as "exception -> HRESULT -> exception -> HRESULT -> exception".

Unfortunately, there is no standard COM facility to pass call stack from callee to caller. It's not possible to pass original exception's information to the caller. You may try different approaches as explained in this article 467 (see "Creating bug reports for DLL exceptions" section at the end of the article).

Using COM objects with exception tracer without framework (VCL)

There are no any special issues if you write COM server manually without using VCL support. It would be just a normal DLL with exported functions. Consider COM rules as system API. **You have to wrap each method in try/except block and invoke EurekaLog to handle exceptions.**

Note: "Handle every SafeCall exception" option 341 will have no effect, since you're not using `ComObj` unit. You may use `safecall` methods instead of stdcall/HRESULT - based on `TObject`. However, you still need to manually invoke EurekaLog from your SafeCallException override.

Hints about options for your COM server project without VCL:

- Application profile 234 should be "Standalone DLL";
- (Optional) If you're going to use VCL, CLX or FireMonkey forms in your COM object: enable hooks for corresponding framework 352;
- Since exceptions in COM objects are handled in the same module (even more: in the same method) - you should enable "Capture stack only for exceptions from current module" option 237;
- Switch dialog to "None" 267;
- Multi-threaded COM objects: set "Default EurekaLog state in new threads" option 246 to "Enabled for RTL threads, disabled for Windows threads" value or enable EurekaLog manually 547 for each thread.

See for more information:

- Creating bug reports for DLL exceptions 470
- Using exception tracer tool in DLLs 480
- Working with system or 3rd party API 459
- Using EurekaLog in multi-threaded applications 547

Using COM objects with exception tracer and framework (VCL)

This approach is much easier, because you don't have to write a lot of code manually. You don't have to write any try/except blocks. You only have to enable "Handle every SafeCall exception" option 341.

Short summary of configuration:
- Application profile 234 should be "Standalone DLL";
- (Optional) If you're going to use VCL, CLX or FireMonkey forms in your COM object: enable hooks for corresponding framework 352;
- Since exceptions in COM objects are handled in the same module (even more: in the same method) - you should enable "Capture stack only for exceptions from current module" option 237;
- Switch dialog to "None" 267;
- Multi-threaded COM objects: set "Default EurekaLog state in new threads" option 246 to "Enabled for RTL threads, disabled for Windows threads" value or enable EurekaLog manually 547 for each thread;
- Enable "Handle every SafeCall exception" option 341.
- (Optional) Enable "Fix TObject.SafeCallException for hardware exceptions" option 352.
- You must do at least one of the following:
  - o Enable "Use low-level hooks" option 259;
  - o Enable "Enable extended memory manager" option 250;
  - o Use Delphi 2009 or above.

You can also customize EurekaLog behavior via event handlers 192. For example, you can disable dialog for `safecall` exceptions, but enable dialog for exceptions in forms. Note that `safecall` exceptions are considered to be handled exceptions.

Note: you don't have to enable "Catch handled exceptions" option 341.

You can also study COM object demo application shipped with EurekaLog installation.

See also:
- Creating bug reports for DLL exceptions 470
- Using exception tracer tool in DLLs 480
- Working with system or 3rd party API 459
- Developing DLL API 467
- Using EurekaLog in multi-threaded applications 547

11.4.6 Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.)

EurekaLog supports reading of some 3rd party formats of debug information. This feature could be used in a migration scenario: when you migrate your multi-DLL project from other solution (such as JCL, madExcept, etc.) to EurekaLog. You can re-use old DLLs without recompiling these DLLs for EurekaLog. Your application should use the single exception tracer scheme 474.

1. Your host (.exe file) should be EurekaLog-enabled with enabled support for 4rd party debug information formats (see below).
2. Your DLL files could be:
   - o EurekaLog-enabled DLLs (using "DLL profile" 368)
     OR
   - o DLLs with 3rd party debug information which were post-processed by 3rd party compilers (such as JCL, madExcept, etc.)

You can mix EurekaLog-enabled DLLs and "3rd party-enabled DLLs" in the same application. In other words, EurekaLog-enabled DLLs with "DLL" profile (i.e. without exception tracer in DLL) are interchangeable with DLLs post-processed by 3rd party tools (without including a working exception tracer in DLL). "Standalone DLL" profile is not compatible with 3rd party exception tracers.

Host application should have ability to read debug information from DLLs. EurekaLog supports many formats of debug information 409. Support for EurekaLog's own format of debug information is always enabled. Other formats should be enabled manually in

EurekaLog project's options 355.

Note: there is no support to convert debug information from 3rd party post-processor tools into EurekaLog debug information format. That's because all such formats are very similar to each other. There is no significant benefit from converting debug information from some debug information format to another format. Therefore you should just enable support for particular format in your application. No convertation is necessary.

See also:

- Single instance of exception tracer 474
- Debug information providers 409
- Debug information providers configuration 355

11.4.7 Using EurekaLog with non-Embarcadero DLLs

EurekaLog can be used with DLLs compiled by non-Embarcadero compilers - such as Microsoft Visual Studio, etc. 3rd party compiler must generate debug information in some of supported by EurekaLog formats (see list 409). Your application should use the single exception tracer scheme 474.

Note: PDB format is a modern debug information format for Microsoft Visual Studio tool chain. It can contain much more information than older DBG debug information format. DBG format support is limited in many tools. For example. It is recommended to use PDB format when possible.

You can mix EurekaLog-enabled DLLs and 3rd party compilers DLLs in the same application. In other words, EurekaLog-enabled DLLs with "DLL" profile (i.e. without exception tracer in DLL) are interchangeable with DLLs compiled by 3rd party compiler for all of the 3 cases above (i.e. converting debug information to EurekaLog format, enabling support for additional formats, or using plain DLL exports information). "Standalone DLL" profile is not compatible with 3rd party exception tracers.

For the purposes of this article we will use the following sample code:

MSSample.cpp file:

```
#include "stdafx.h"
#include "MSSample.h"

void InternalTest(void)
{
   int * P;
   P = NULL;
   *P = 0;
}

void Test(void)
{
   InternalTest();
}

MSSAMPLE_API int fnMSSample(void)
{
   Test();
   return 42;
}
```

MSSample.h file:

```
#ifdef MSSAMPLE_EXPORTS
```

```
#define MSSAMPLE_API __declspec(dllexport)
#else
#define MSSAMPLE_API __declspec(dllimport)
#endif

MSSAMPLE_API int fnMSSample(void);
```

Unit1.pas file:

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TTestProc = function: Integer; cdecl;
var
  Lib: HMODULE;
  Test: TTestProc;
begin
  Lib := LoadLibrary('MSSample.dll');
  Win32Check(Lib <> 0);
  try
    try
      Test := GetProcAddress(Lib, '?fnMSSample@@YAHXZ');
      Win32Check(Assigned(Test));
      Test;
    except
      Application.HandleException(Sender);
    end;
  finally
    FreeLibrary(Lib);
  end;
end;
```

This sample DLL is compiled by Microsoft Visual Studio. It contains one exported function (fnMSSample) which calls some internal functions (Test and InternalTest) and raises access violation exception. It is loaded and called by the Delphi project (.exe host).

You should enable EurekaLog for host application. You should do this in the same way as you do it for typical application without any DLLs. For example, if you have VCL Forms application as the host - then you need to enable EurekaLog for host application and set application type to "VCL Forms Application". This will add EurekaLog code and data into final .exe file. It would also set hook for `Forms.TApplication.HandleException` method, which will allow to automatically handle exceptions without writing any code.

Important note: Please note that above example is not recommended to use in real projects. The example above is created to illustrate differences in debug information for DLLs. It is not intended to illustrate DLL design principles. The problem with example design is as following: the sample DLL does not contain any try/catch handlers, and it lets exceptions escape DLL to the caller. This is usually a bad practice 457 - because the caller may not know how to work with exceptions coming from other programming language. The above example uses hardware exception for illustration. Real-life application will probably raise software exceptions - which are specific to programming language. Thus, a better idea is to wrap fnMSSample function into try/catch block and convert exception into safe error code (a simple flag, integer code, HRESULT, etc.) - as explained in this article 467.

Since we're going to pass exceptions from DLL to .exe (a not recommended way, but sufficient for our example) - you have to do the following:

- Disable "Capture stack only for exceptions from current module" option 237. This will instruct EurekaLog to catch exceptions from any executable module. By default EurekaLog captures only exceptions within the same module.
- Disable chained exceptions support 573 by setting all options to "Classic" position. This feature requires ability to track life time of exceptions objects. This is not possible for general case (e.g. host and DLL are compiled by different compilers and there is no assist

from RTL for tracking exception objects). This feature *may* work in some specific configurations.

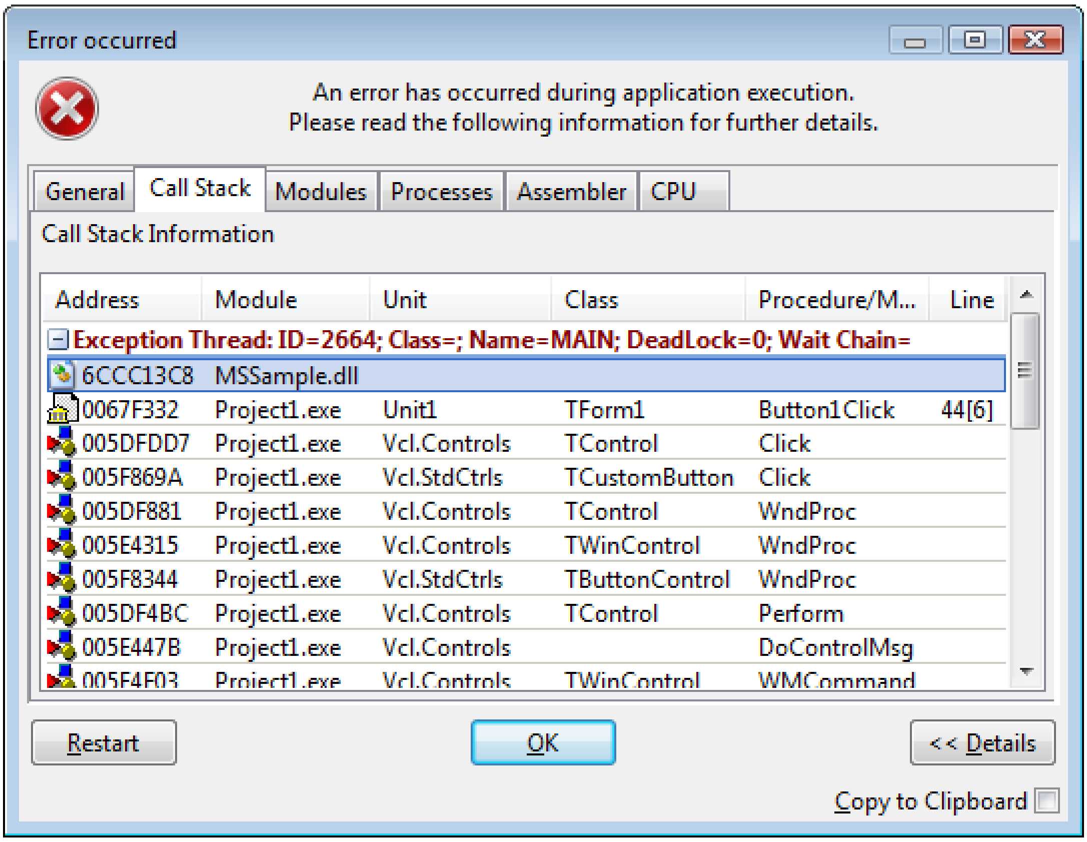
You don't need to perform these changes if you're using the recommended approach 457 of not letting exceptions escape DLL. See example here 467.

Note: notice that event handler for Button1 in the above example calls exception handler (Application.HandleException) explicitly. This is a required action for such code. That is because exceptions from DLL will be handled in default application handler without such explicit call - which happens after DLL will be unloaded. Therefore, an execution will go such way without explicit try/except block:

- Load DLL
- Call function from DLL
- Raise exception
- Unload DLL
- Analyze exception from (already unloaded) DLL

The last step will fail because DLL was already unloaded. This is the reason behind explicit call to Application.HandleException. Please note that you don't need such call if you use different buttons to load DLL/call function/unload DLL - that is because default exception handler will guard each call of event handler.

This sample will generate the following error dialog without any additional actions:



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/M...	Line
Exception Thread: ID=2664; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll				
0067F332	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005DFDD7	Project1.exe	Vcl.Controls	TControl	Click	
005F869A	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005DF881	Project1.exe	Vcl.Controls	TControl	WndProc	
005E4315	Project1.exe	Vcl.Controls	TWinControl	WndProc	
005F8344	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005DF4BC	Project1.exe	Vcl.Controls	TControl	Perform	
005E447B	Project1.exe	Vcl.Controls		DoControlMsg	
005F4F03	Project1.exe	Vcl.Controls	TWinControl	WMCommand	

Exception dialog for DLL without any debug information support

As you can see: the call stack lists every function within host .exe file - because host .exe has EurekaLog debug information. DLL have no debug information. Therefore, EurekaLog is unable to show call stack for DLL. Empty first line is exact exception address. It is shown always - regardless of debug information available.

Of course, this is not a very useful bug report for DLL. You want to see some functions from that DLL.

There are three possible usage cases for non-Embarcadero DLLs:

1. Convert 3rd party debug information format into EurekaLog debug information;
2. Use 3rd party debug information without converting to EurekaLog debug information;
3. Use DLL exports provider (no debug information is available or debug information format is not supported).

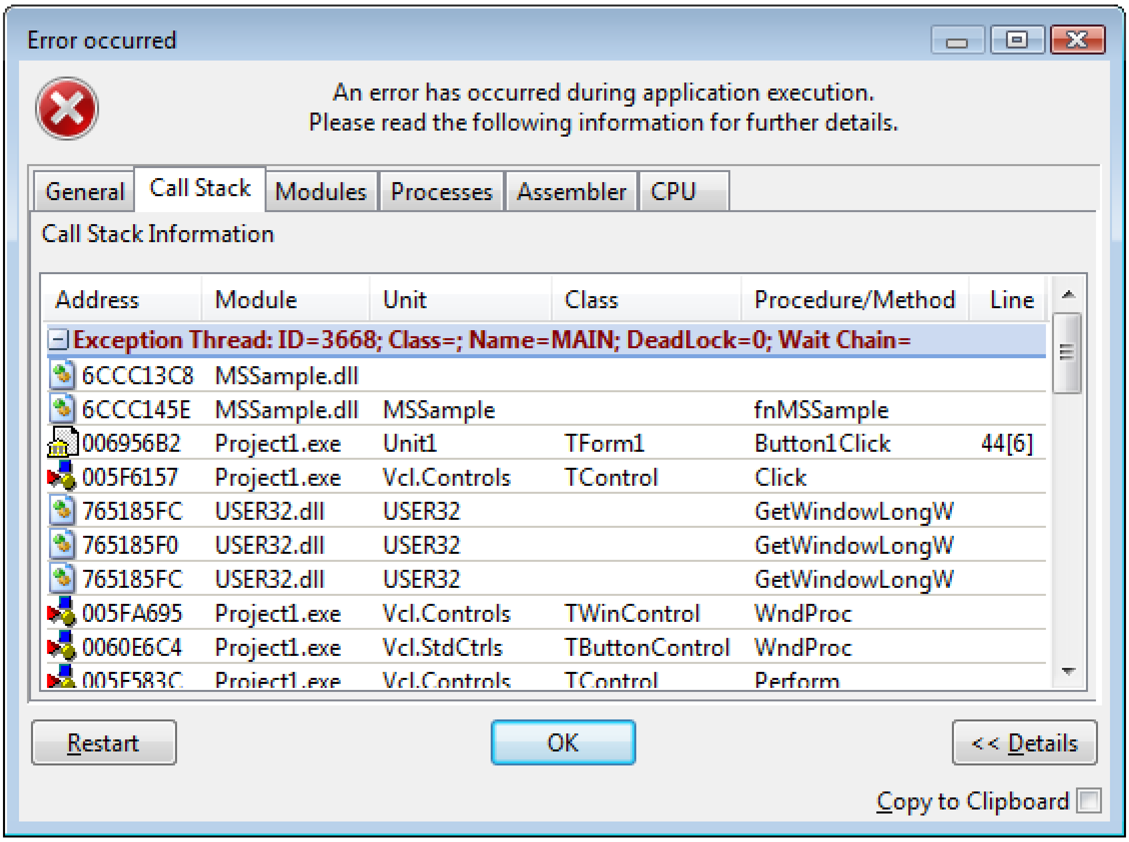
Use DLL exports provider (no debug information is available or debug information format is not supported)

This approach is not recommended - because there will be inaccurate information in call stacks without debug information. This case should be used only if compiler is not able to produce debug information or debug information format is unknown to EurekaLog. Use this approach as "last resort" measure: to show at least something for DLLs without debug information.

1. Enable DLL exports debug information provider 355 (see description of this provider 411).

Note: this case is a default configuration for EurekaLog.

The result error message will look like this:



Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Exception Thread: ID=3668; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll				
6CCC145E	MSSample.dll	MSSample		fnMSSample	
006956B2	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F6157	Project1.exe	Vcl.Controls	TControl	Click	
765185FC	USER32.dll	USER32		GetWindowLongW	
765185F0	USER32.dll	USER32		GetWindowLongW	
765185FC	USER32.dll	USER32		GetWindowLongW	
005FA695	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060E6C4	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F583C	Project1.exe	Vcl.Controls	TControl	Perform	

Copy to Clipboard □

Exception dialog for DLL without debug information but with DLL Export provider

This example is able to discover name of exported function ('fnMSSample') - thanks to DLL Export provider. However, this example is not able to identify internal functions in DLL - because internal functions are not exported. Therefore, internal functions are not listed at all. And (as always) exact exception location is added to the top of the call stack.

Please note that this example also adds entries for USER32.dll and KERNEL32.dll in the call stack.

Note: DLL Exports provider may show entries like "(possible fnMSSample+132)". Such text means that there are some JMP or RET instructions between start of the function and actual address in a call stack. This means:

- [Positive] Address belongs to the specified function. JMP/RET instruction may be part of the function's logic (such as try/except block);

- [False-positive] Address does not belong to the specified function. JMP/RET instruction marks the end of the function. Address itself lies within some other internal/unknown function after the specified function.

Number after "+" sign indicate byte offset between function's start and call stack address. Greater offsets usually indicate greater chance for false-positive entries.

Use 3rd party debug information without converting to EurekaLog debug information

This approach can be used if you want to use other tools for your executable (for example: Process Explorer, WinDBG, external debugger, etc.). Other tools are not able to recognize and read EurekaLog debug information. Thus, you need to supply and keep debug information in a known format - such as PDB/DBG, TD32, etc. Both EurekaLog and other tools will be able to use this debug information.

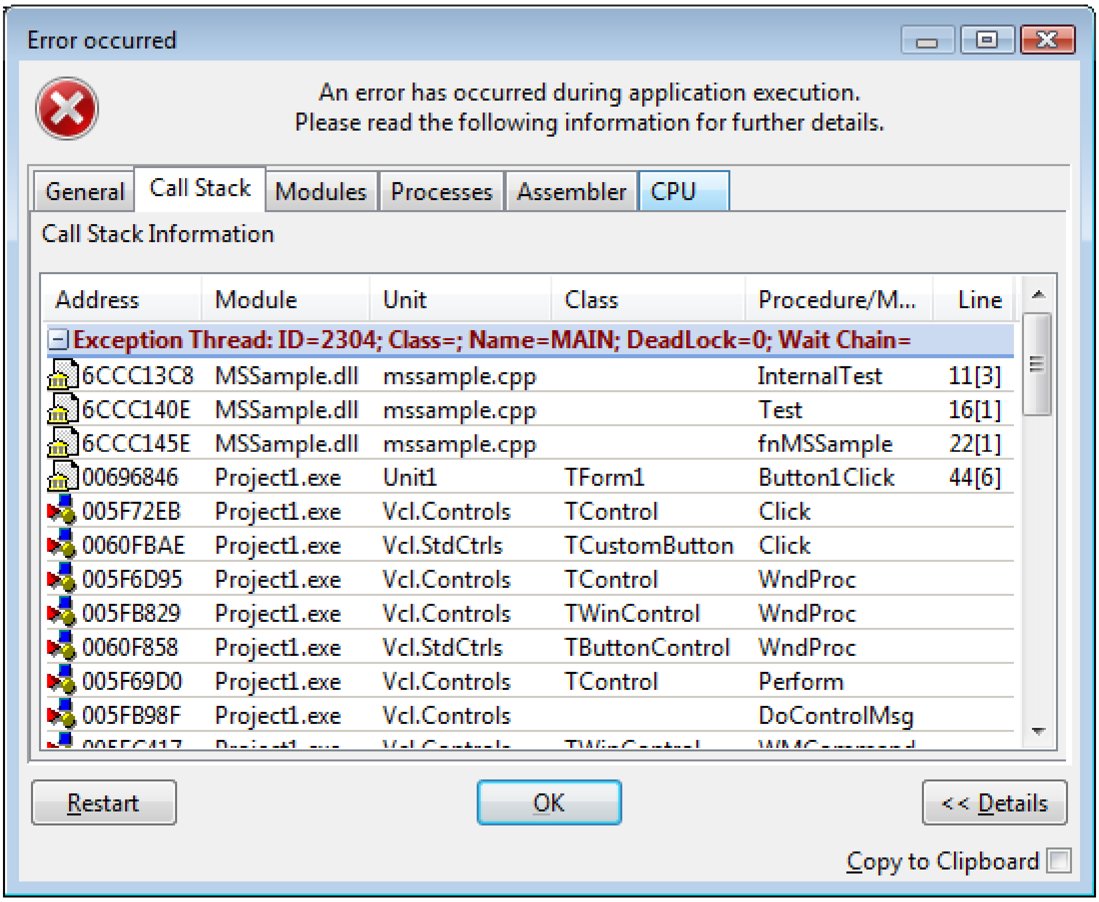
1. Enable generation of debug information in project's options (see below);
2. Enable support for debug information format 355 in EurekaLog's project options (see list of supported formats 409).

For example, we use Microsoft Visual Studio in the above example. You can go to "Project" / "Properties" IDE menu item to open options for your C++ DLL project. Go to "Configuration Properties" / "Linker" / "Debugging" and enable "Generate debug info" option. Go to Go to "Configuration Properties" / "C/C++" / "General" and set "Debug Information Format" option option to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler). Build your project. There should be .pdb file available in the same folder as .dll file for your project.

You should deploy this .pdb file with your .dll file.

Enable support for PDB debug information format by enabling "Microsoft Dbg/PDB" option 355 .

Note: you can enable generation of .map files in your Visual Studio projects. However, such files can not be used by EurekaLog. .map files do not have a strict format. .map files are defined as "human-readable plain text files in free form that indicate the relative offsets of functions for a given version of a compiled binary". EurekaLog is able to parse .map files produced by Delphi and C++ Builder linkers. EurekaLog is not able to parse .map files produced by other compilers/linkers/tools.



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/M...	Line
⊟Exception Thread: ID=2304; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll	mssample.cpp		InternalTest	11[3]
6CCC140E	MSSample.dll	mssample.cpp		Test	16[1]
6CCC145E	MSSample.dll	mssample.cpp		fnMSSample	22[1]
00696846	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F72EB	Project1.exe	Vcl.Controls	TControl	Click	
0060FBAE	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005F6D95	Project1.exe	Vcl.Controls	TControl	WndProc	
005FB829	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060F858	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F69D0	Project1.exe	Vcl.Controls	TControl	Perform	
005FB98F	Project1.exe	Vcl.Controls		DoControlMsg	

Exception dialog for DLL with .pdb file and enabled MS Debug Info provider

Since DLL now have full debug information available in .pdb file, and EurekaLog has enabled support for reading .pdb files - there will be full information for your DLL in the call stack. All exported and internal functions will be properly identified. All functions will have line numbers information.

Note:
- **IMPORTANT: (only for "Microsoft Dbg/PDB" provider)** You have to deploy .pdb file with your DLL. Unfortunately, PDB information can not be injected into executable. Only standalone .pdb files are supported. This is not a limitation of EurekaLog;
- **IMPORTANT: (only for "Microsoft Dbg/PDB" provider)** You have to deploy DbgHelp.dll file from Microsoft Debugging Tools. This file can also be found in \Bin (\Bin64) folder of EurekaLog installation. Default DbgHelp.dll from C:\Windows is not suitable for such usage.
- You can use any other source of debug information: such as TD32 (.tds), MAP (.map), DBG (.dbg), JDBG (.jdbg), etc. Just be sure to enable corresponding debug information provider 355. Most other debug formats can be injected into executable and not require any helper DLLs;
- You can use PE Analyzer (Module Informer) tool 617 to check whenever DLL has any supported debug information;
- You can also enable debug information for Windows DLLs to show precise information about internal functions in system DLLs. This configuration is explained here 504.

Convert 3rd party debug information format into EurekaLog debug information

.pdb files are analog of .tds files (with TD32 debug information): these files are extremely large (debug information file could be more than 10x times larger than executable module itself), binary, uncompressed, unencrypted, store huge amount information about executable (functions, arguments, types, classes, scopes, line numbers, etc. - in other words, all information that may be needed for the debugger). Since exception tracer does not need all this information (units/routine names and line numbers are enough) - obviously, deploying such files along with your DLLs is not a best solution. Surely, you have to use .pdb files if you need to load your DLL into other tools (such as Process Explorer or WinDBG), but if you just want to use exception tracer tool with your DLL - there must be a better way.

A better way is to convert 3rd party debug information into EurekaLog debug information. EurekaLog debug information is compact, compressed, encrypted and stores only minimum amount of information necessary to build call stack. All other extra information is not stored. And you won't need any external helper DLLs - like DbgHelp.dll.

This is recommended approach. You can convert some supported debug information formats into EurekaLog debug information format. You can do this without DLL recompilation 426.

1. Enable generation of debug information in project's options (the same as in the previous approach - see above);
2. Compile your DLL. There will be .dll file and debug information file (such as .pdb);
3. Run ecc32.exe or emake.exe to post-process your DLL file with embedding EurekaLog debug information 38 (see below).

Ecc32/emake tools can use --el_alter_exe command line switch to specify target .dll file for post-processing (you should use NUL as project file name for --el_alter_exe switch - since your DLL is not a Delphi / C++ Builder project), --el_config switch to specify EurekaLog configuration (you have to use external .eof file since there is no Delphi / C++ Builder project to read configuration from), --el_source to specify debug information source (default source is Delphi / C++ Builder .map files; you have to specify where ecc32/emake should look for debug information).

Notes:
- You have to create new .eof file 443 which will contain EurekaLog configuration for your DLL. This file is required since there is no Delphi / C++ Builder project (which usually stores EurekaLog configuration).
- Most options in .eof file will be ignored since there is no EurekaLog code in your DLL. Only design-time / build options will be used: such as encryption for debug information, stripping relocs, removing function names, etc.
- You can enable generation of .map files in your Visual Studio projects. However, such files can not be used by EurekaLog. .map files do not have a strict format. .map files are defined as "human-readable plain text files in free form that indicate the relative offsets of functions for a given version of a compiled binary". EurekaLog is able to parse .map files produced by Delphi and C++ Builder linkers. EurekaLog is not able to parse .map files produced by other compilers/linkers/tools. Therefore, only Delphi / C++ Builder .map files could be used for post-processing. Other possible source includes .tds files (TD32), any format that DbgHelp supports (usually: .pdb and .dbg files).
- PDB format is a modern debug information format for Microsoft Visual Studio tool chain. It can contain much more information than older DBG debug information format. DBG format support is limited in many tools. For example. It is recommended to use PDB format when possible.

For example, we have Microsoft Visual Studio DLL project as described above. Project have some internal function and exports one function (fnMSSample). You can go to "Project" / "Properties" IDE menu item to open options for your C++ DLL project. Go to "Configuration Properties" / "Linker" / "Debugging" and enable "Generate debug info" option. Go to Go to "Configuration Properties" / "C/C++" / "General" and set "Debug Information Format" option option to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler). Build your project. There should be .pdb file available in the same folder as .dll file for your project.

Now, run ecc32.exe or emake.exe with the following command-line:

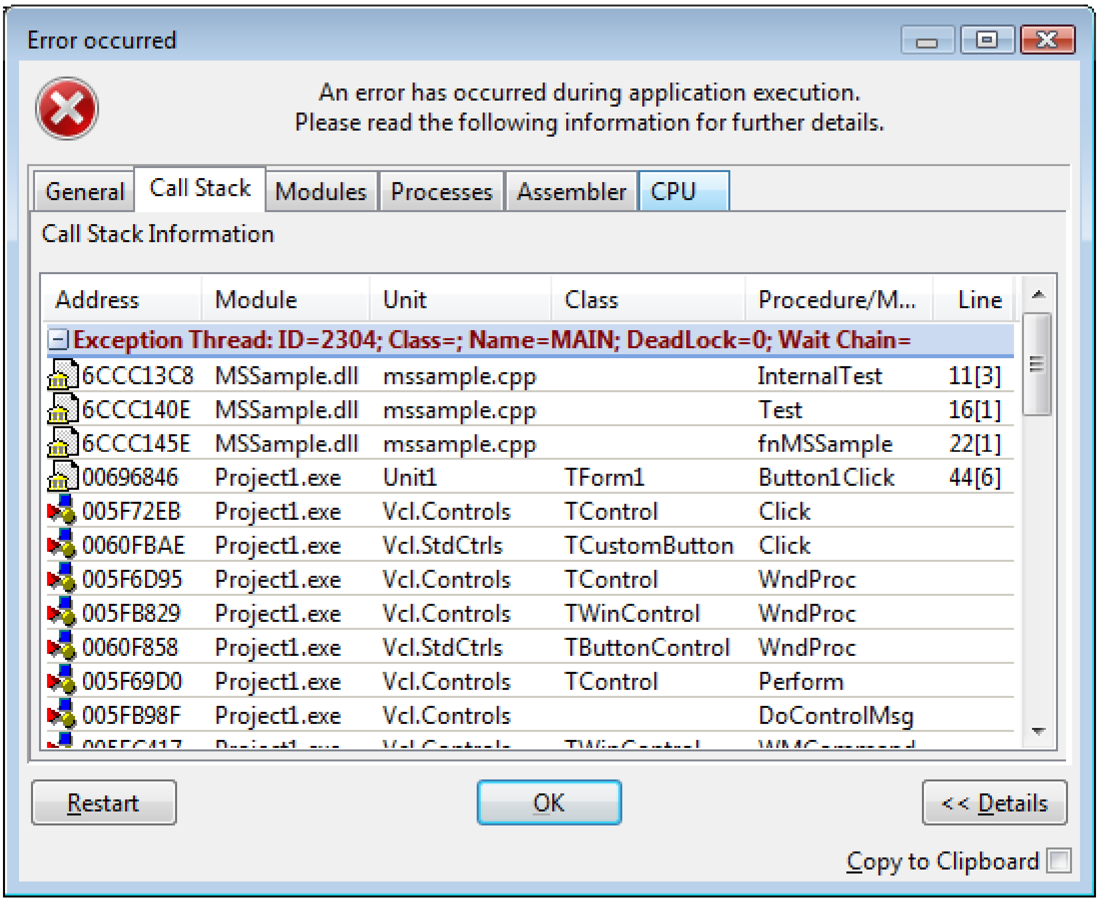
```
"ecc32.exe" "--el_alter_exe=NUL;MSSample.dll" "--el_config=MSSample.eof" --el_source=PDB
```

(you may need to specify full or relative file paths for your files; do not forget to enclose file paths with spaces in double quotes)

This command line will convert MSSample.pdb file into EurekaLog format and inject this information into MSSample.dll file. Options for this operation are specified in MSSample.eof file.

Note: MSSample.pdb file may be deleted after conversion - depending on the state of "Delete service files after compilation" option 234.

Resulting DLL will have injected EurekaLog debug information - which could be verified by using PE Analyzer (Module Informer) tool 617. No additional debug information providers should be enabled. The exception dialog will look like this (the same as in the previous approach):



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/M...	Line
⊟ Exception Thread: ID=2304; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll	mssample.cpp		InternalTest	11[3]
6CCC140E	MSSample.dll	mssample.cpp		Test	16[1]
6CCC145E	MSSample.dll	mssample.cpp		fnMSSample	22[1]
00696846	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F72EB	Project1.exe	Vcl.Controls	TControl	Click	
0060FBAE	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005F6D95	Project1.exe	Vcl.Controls	TControl	WndProc	
005FB829	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060F858	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F69D0	Project1.exe	Vcl.Controls	TControl	Perform	
005FB98F	Project1.exe	Vcl.Controls		DoControlMsg	
005FC417	Project1.exe	Vcl.Controls	TWinControl	WMCommand	

Exception dialog for DLL with injected EurekaLog debug information

This call stack will be the same as in the previous approach: DLL now have full debug information which is injected into DLL file - there will be full information for your DLL in the call stack. All exported and internal functions will be properly identified. All functions will have line numbers information.

The difference from the previous approach is that you don't need to deploy any additional files along with your DLL. All necessary debug information is stored inside DLL file itself.

See also:

- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
- Post-processing without (re)compilation 426
- Working with configuration 439
- External configuration and .eof files 443
- PE Analyzer (Module Informer) tool 617

11.4.8 Using Microsoft DbgHelp DLL

You can use MS debug format[412] to get information for system DLLs. By default Windows comes in "release" version - i.e. without debug information for system DLLs (so-called "free build"). This prevents you from getting proper information. In fact, you only can get heuristic information based on DLL exports[411]. However, you can ask Microsoft for debug information and get full coverage for Microsoft DLLs.

Basically, when your application needs debug information about system DLL - it can ask Microsoft server for specific version of system DLL and download corresponding debug information file. Unfortunately, this process is not well-suited for end-users machines, but acceptable for developers machines. This information can be used not only by EurekaLog, but also by other tools.

1. You'll need a good internet connection.
2. A lot of free disk space (about 512 Mb).
3. You need to create a writable folder which will be used as local cache.
4. You have to use the latest DbgHelp.dll. Default DLL which ships with Windows will not work. Use DLL either from Microsoft Debugging Tools for Windows or from \Bin (\Bin64) folders of EurekaLog installation.
5. You have to setup retrieving debug information (see below).

Warning: using this process will slow down building call stacks and bug reports in your application - because information will be downloaded from the internet. This is especially true for the first time use, when many information needs to be downloaded and it not present in the cache. Therefore you should disable hang detection in your application.

A sample call stack **without** using debug information for system DLLs (information was provided by DLL exports provider[411]):

```
| Module     | Unit       | Class Name | Routine Name                    | Line   |
|richedit.exe|remain      |            |Enumer                           |216[2]  |
|USER32.dll  |USER32      |            |(possible GetWindow+508)         |        |
|USER32.dll  |USER32      |            |EnumWindows                      |        |
|richedit.exe|remain      |TMainForm   |UpdateStatus                     |225[1]  |
|richedit.exe|remain      |TMainForm   |SetFileName                      |234[4]  |
|richedit.exe|remain      |TMainForm   |PerformFileOpen                  |325[3]  |
|richedit.exe|remain      |TMainForm   |FileOpen                         |361[4]  |
...
|USER32.dll  |USER32      |            |(possible GetThreadDesktop+296)  |        |
...
|richedit.exe|Classes     |            |StdWndProc                       |        |
|USER32.dll  |USER32      |            |(possible gapfnScSendMessage+818)|        |
|USER32.dll  |USER32      |            |(possible GetThreadDesktop+128)  |        |
|USER32.dll  |USER32      |            |(possible CharPrevW+307)         |        |
|USER32.dll  |USER32      |            |DispatchMessageW                 |        |
|richedit.exe|Forms       |TApplication|HandleMessage                    |        |
|richedit.exe|Forms       |TApplication|Run                              |        |
|richedit.exe|richeditdemo|            |Initialization                   |24[4]   |
|kernel32.dll|kernel32    |            |BaseThreadInitThunk              |        |
```

As you can see - entries from User32.dll are confusing. Some of them mentions only "possible" match and this match is usually wrong. This is the best that you can do without debug information.

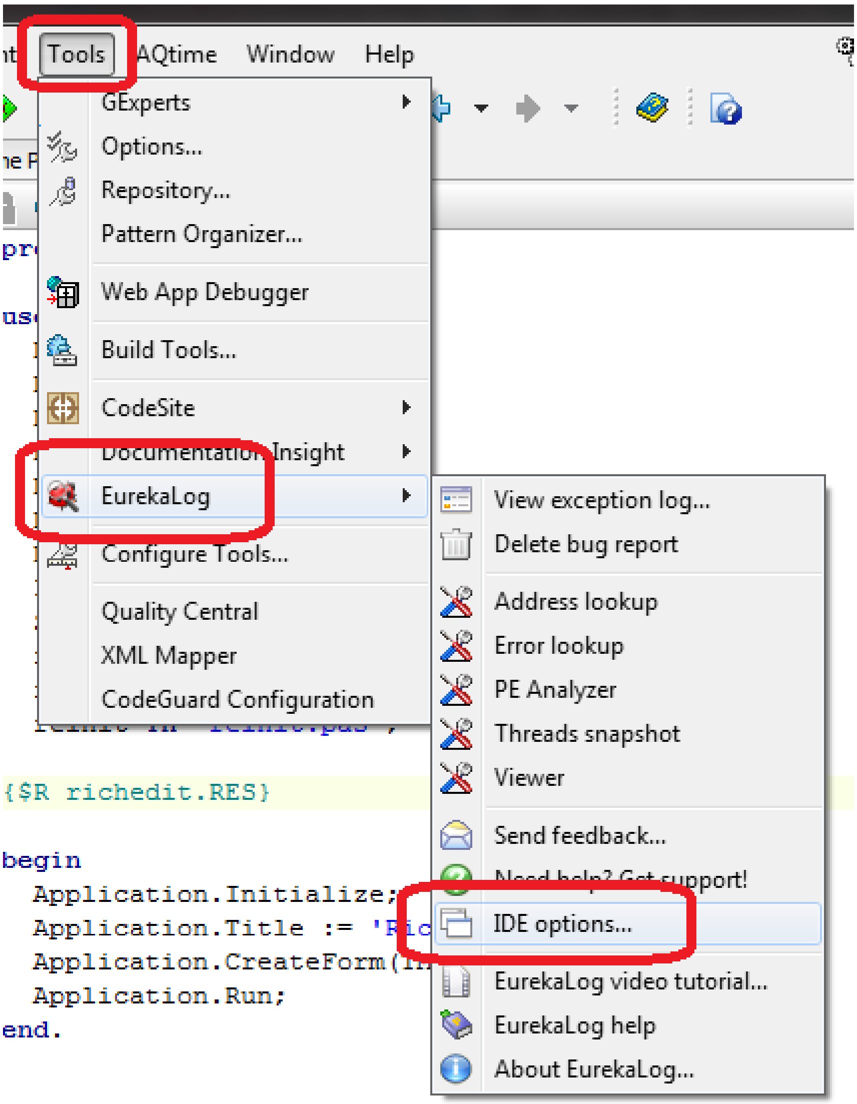
Consider the same call stack when retrieving debug information **is set** (information was provided by Microsoft provider[412]):

```
| Module     | Unit       | Class Name | Routine Name          | Line   |
|richedit.exe|remain      |            |Enumer                 |216[2]  |
|USER32.dll  |USER32      |            |InternalEnumWindows    |        |
|USER32.dll  |USER32      |            |EnumWindows            |        |
|richedit.exe|remain      |TMainForm   |UpdateStatus           |225[1]  |
|richedit.exe|remain      |TMainForm   |SetFileName            |234[4]  |
|richedit.exe|remain      |TMainForm   |PerformFileOpen        |325[3]  |
|richedit.exe|remain      |TMainForm   |FileOpen               |361[4]  |
...
|USER32.dll  |USER32      |            |CallWindowProcAorW     |        |
...
|richedit.exe|Classes     |            |StdWndProc             |        |
|USER32.dll  |USER32      |            |InternalCallWinProc    |        |
|USER32.dll  |USER32      |            |UserCallWinProcCheckWow|        |
|USER32.dll  |USER32      |            |DispatchMessageWorker  |        |
|USER32.dll  |USER32      |            |DispatchMessageW       |        |
|richedit.exe|Forms       |TApplication|HandleMessage          |        |
|richedit.exe|Forms       |TApplication|Run                    |        |
|richedit.exe|richeditdemo|            |Initialization         |24[4]   |
|kernel32.dll|kernel32    |            |BaseThreadInitThunk    |        |
```

As you can see - entries from User32.dll are now correct.

Enabling downloading debug information for developer machine

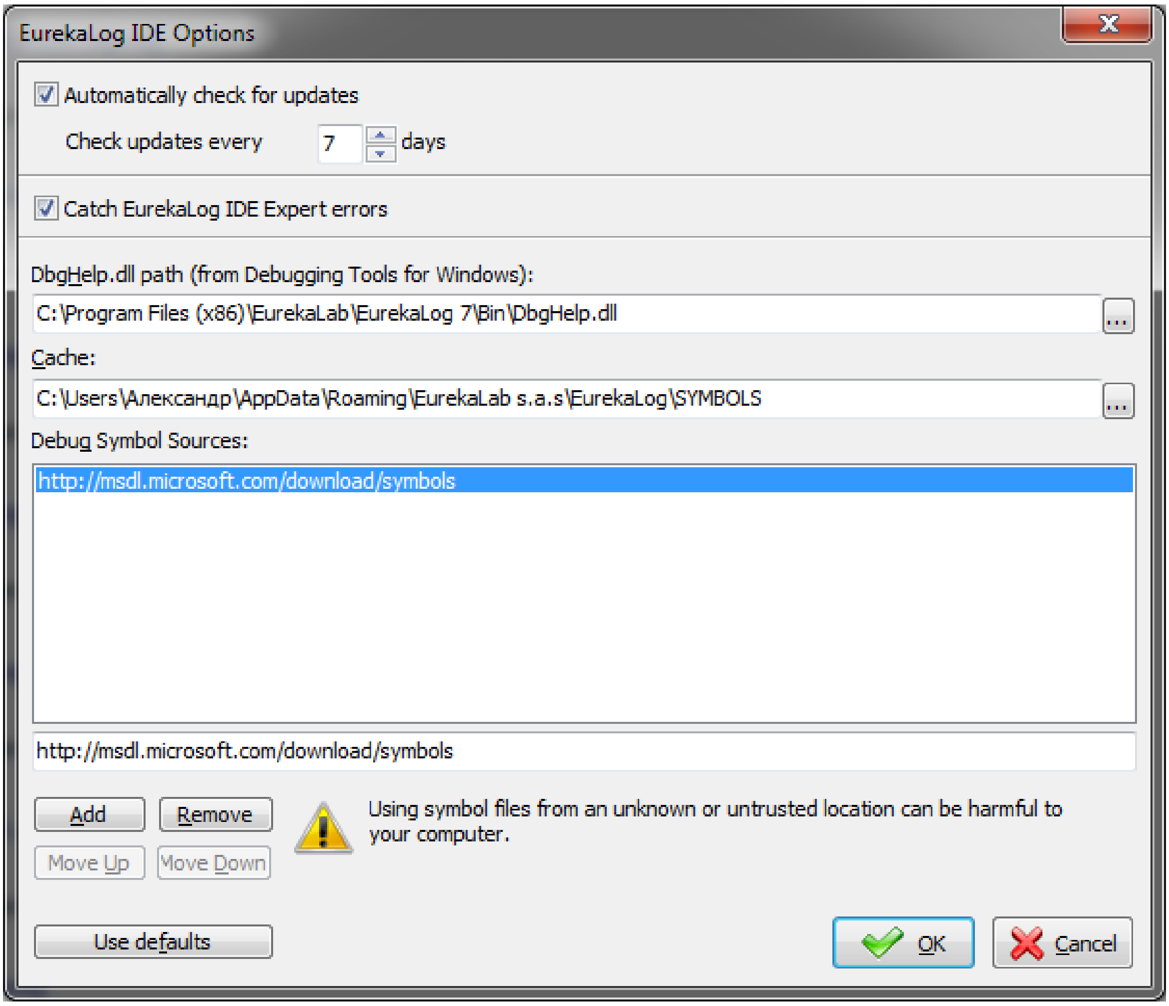
You can enable this feature by going to "Tools" / "EurekaLog" / "IDE options" menu item:



```
{$R richedit.RES}

begin
  Application.Initialize;
  Application.Title := 'Ric
  Application.CreateForm(T
  Application.Run;
end.
```

Opening global EurekaLog options

You'll see a dialog to set global EurekaLog options 230:



EurekaLog IDE Options

☑ Automatically check for updates

Check updates every 7 days

☑ Catch EurekaLog IDE Expert errors

DbgHelp.dll path (from Debugging Tools for Windows):

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Bin\DbgHelp.dll

Cache:

C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\SYMBOLS

Debug Symbol Sources:

http://msdl.microsoft.com/download/symbols

http://msdl.microsoft.com/download/symbols

Using symbol files from an unknown or untrusted location can be harmful to your computer.

EurekaLog IDE options

1. "**DbgHelp.dll path**" option specifies full file name to DbgHelp DLL. You can find it in EurekaLog's installation folder (like: `C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Bin\dbghelp.dll`). You can also change this path to alternative library. This is DLL from Debugging tools for Microsoft Windows.

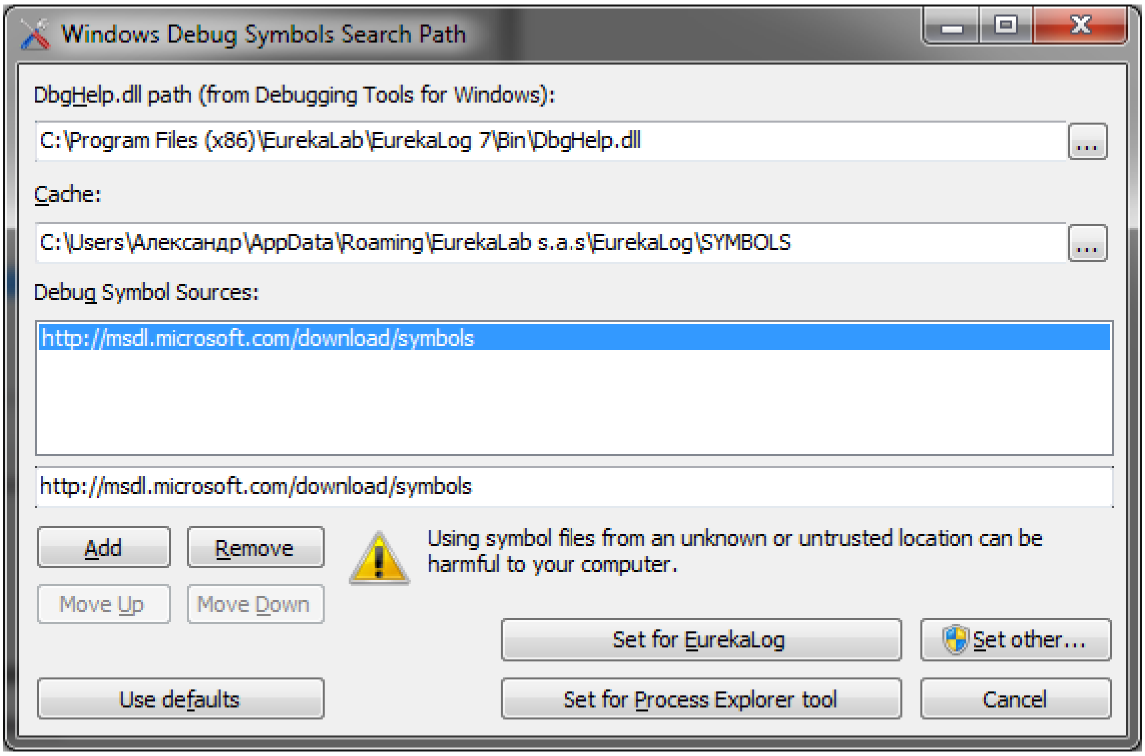
2. "**Cache**" option specifies read-write folder to be used as cache for debug info symbols. It's empty by default, which means disabled feature. You can click on "**Use defaults**" button to set default preference (which is sub-folder in your %APPDATA%, for example: `C:\Users\`*`User`*`\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\SYMBOLS`) or select your own folder. Be sure that this folder is writable and disk have some free space (500 Mb minimum). This cache folder is used by MS Debug info provider 412 to store debug symbols for system libraries.

3. "**Debug symbol sources**" option specifies debug info sources for MS Debug info provider 412. You can add one or more sources here by using edit-box and buttons below. Source can be local folder (like: C:\Symbols), shared network path (like: \\server\symbols) or URL of symbol's server (like: http://server/symbols). It's empty by default, which means disabled feature. You can click on "**Use defaults**" button to set default preference (which is default Microsoft's symbol server: `http://msdl.microsoft.com/download/symbols`) or select your own sources.

To quickly enable feature - you can just click on "**Use defaults**" button and close options dialog via **OK** button. To disable feature - remove all debug symbol sources.

Enabling downloading debug information for other machines

To enable or disable this feature on any other machine you can use "Set Debug Symbols Path" tool, which can be found in \Bin\SetDebugSymbolsPath.exe under EurekaLog installation folder. This tool can be copied on another machine and used to set up path to DLL, to cache, and to debug symbols source. It has the same UI as IDE options dialog:



DbgHelp.dll path (from Debugging Tools for Windows): C:\Program Files (x86)\EurekaLab\EurekaLog 7\Bin\DbgHelp.dll

Cache: C:\Users\Александр\AppData\Roaming\EurekaLab s.a.s\EurekaLog\SYMBOLS

Debug Symbol Sources: http://msdl.microsoft.com/download/symbols

http://msdl.microsoft.com/download/symbols

Using symbol files from an unknown or untrusted location can be harmful to your computer.

Set Debug Symbols Path Tool

To enable feature - select DbgHelp.dll, specify path to writable folder, add URL, and click on "**Set for EurekaLog**" button. To disable feature - remove all debug symbol sources and click on "**Set for EurekaLog**" button.

Enabling downloading debug information for other tools (non-EurekaLog)

You can also use the similar feature in other debugging software. You can use "Set Debug Symbols Path" tool to setup this feature or you can do it manually. To setup it with "Set Debug Symbols Path" tool - copy \Bin\SetDebugSymbolsPath.exe file to another machine, run it, specify all data and click on "Set for Process Explorer tool" to set these settings for Process Explorer tool or "Set other" to set these settings for any other tool. To use "Set other" button you need to run the tool under administrator account.

To configure this feature manually - you need to build "configuration string" in the following format:

SRV**folder***URL*

For example:

SRV*C:\ProgramData\DebugSymbols*http://msdl.microsoft.com/download/symbols

Then you'll need to enter this string into your tool as "Symbols path" folder. Refer to documentation of your tool on where to find this setting. For example, for Process Explorer tool you can find it under "Options" / "Configure symbols" menu item.

Configuring EurekaLog projects to use Microsoft symbols

Any EurekaLog-enabled application can use Microsoft symbols to get information about system functions when building call stack. You can enable this feature by checking "Microsoft DBG/PDB" option in debug information providers configuration 355.

See also:

- Using debug information converters 516
- Using EurekaLog with non-Embarcadero DLLs 496

11.5 Configuring project for leaks detection

This section explains working with memory and resource leaks.

1. Introduction to leaks detection 166;
2. Common scenarios for using leaks detection:
   a. Typical application 512;
   b. Typical DLL 513;
   c. Shared memory manager 513;
   d. Packaged project 514;

11.5.1 About leak detection

While any error in your application is always bad, there are types of errors, which can be not visible in certain environments. For example, memory or resources leaks errors are relatively harmless on client machines and can be deadly on servers.

Memory leaks are a class of bugs where the application fails to release memory when no longer needed. Over time, memory leaks affect the performance of both the particular application as well as the operating system. A large leak might result in unacceptable response times due to excessive paging. Eventually the application as well as other parts of the operating system will experience failures.

Windows will free all memory allocated by the application on process termination, so short-running applications will not affect overall system performance significantly. However, leaks in long-running processes like services or even Explorer plug-ins can greatly impact system reliability and might force the user to reboot Windows in order to make the system usable again.

Applications can allocate memory on their behalf by multiple means. Each type of allocation can result in a leak if not freed after use. Here are some examples of common allocation patterns:

- Allocation via Delphi memory manager wrapper (`GetMem`, `AllocMem`, etc)
- Direct allocations from the operating system via the `VirtualAlloc` function
- Heap memory via the `HeapAlloc` function
- Kernel handles created via Kernel32 APIs such as `CreateFile`, `CreateEvent`, or `CreateThread`, hold kernel memory on behalf of the application
- GDI and USER handles created via User32 and Gdi32 APIs (by default, each process has a quota of 10'000 handles)

Item 1 is called "memory leak" in EurekaLog; items 2-5 are called "resource leak" in EurekaLog.

Why leaks are bad and do I always need to release all memory?

Generally speaking, often mem-leak does not mean any visible problem to a user: application still works. Mem-leaks? So what? Program still do all tasks, that I need from it. This is especially true for client applications: cause they work for a limited amount of time. So mem-leak is not scary - since all memory will be reclaimed at application's exit (refer to Jeffrey Richter's book on native code for more info) and so all leaks will be removed too. No, I don't mean that you don't need to fight mem-leaks here: mem-leak is always bad. It is just that mem-leaks aren't that fatal. Of course, this is not applicable for server applications. Server applications work for long period of time, so even minor leak will be deadly.

Some other question, which is close related to above, is: "if all memory is reclaimed upon app's shutdown - can I skip cleanup for global variables? They still will be deleted by automatic cleanup from OS!"

Well, the formal answer is: "you can do it". This is correct and you really can do it. But "can" does not mean "should". Obviously, there will be no technical problems with that approach. So, why is this bad?

Because you can't find a real mem-leaks, if you do like this. If you don't pedantically clean all of your resources - you will get a bunch of mem-leak reports. Well, leaks "by design": technically it's a leak (since resource wasn't released), but it is not a logical leak. Since you know, that those aren't reports about real mem-leaks - you will ignore them. And the problem is that if there will be a report about real leak - you may just miss it.

That's why it is a common "good rule" to always clean your resources. Unfortunately, there can be cases, when you can't do that. Those are very rare cases, but it can happen. But general rule is: always clean your resources, if you can do it. Don't rely on system's cleanup to throw out garbage for you. This will greatly simplify your life in the future.

Wrong approach

When newbie is inspired of the idea of catching memory leaks - he usually opens the Task Manager and watches "Mem usage" column.

So far so good, but then he suddenly notices that this column behaves very strange, even in a simple application: the memory is not decreasing when closing forms, but decreases at minimizing application, etc, etc. A good question: why does newbie think that this column represents memory allocated by his code? If you open "View"/"Select columns" menu - you'll see many other counters, which also matches "memory definition".

So, I'll open a little secret here: the "Mem Usage" column in Task Manager represents amount of RAM, occupied by your application. This value is not the memory, allocated by your code (you can figure out this by yourself, when you first encounter disappearing of memory at minimizing - of course, no one is going to free your memory without your permission). Your application can use less RAM, then your code allocates, since it can be swapped out to page file. And besides, RAM is spend for code too - namely, for system libraries. System libraries are loaded in every process, but there is only one copy of each DLL in system's memory! (I mean only code here). This value is also called "Working Set". You can see many memory-related values in Task Manager by configuring columns. Or you can use Process Explorer tool (add more columns into process list view too).

So let leave our Task Manager for a while and take a look at Pascal. How Delphi manages its memory? All memory in Delphi application is controlled by memory manager. You can change the memory manager in your application by calling SetMemoryManager. That means that you can detect memory leaks very easy - by installing analyzer stub for memory manager.

What does it mean that your application has a memory leak? Well, this means that your code allocates some memory (object, string, array, etc) and forgets to release it. Forgetting about memory's block means that this memory will still be there at application's exit.

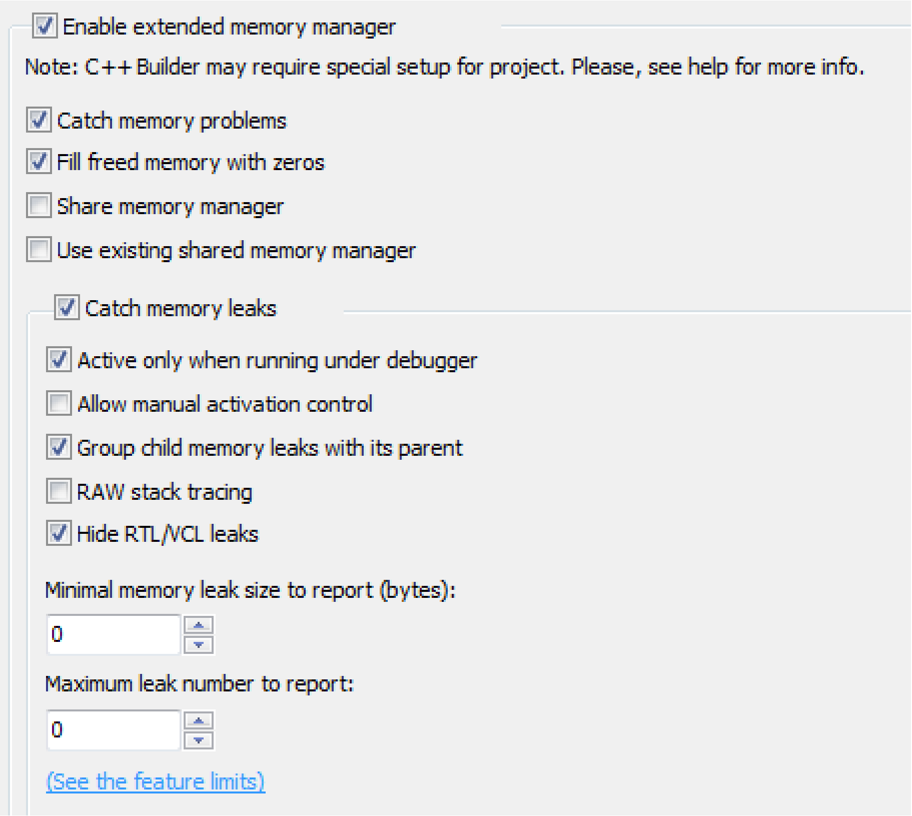
So, to catch all memory leaks you need to enumerate all busy memory blocks at application's exit. Every such block will represent a memory leak.

Using EurekaLog to find memory leaks

EurekaLog has a functionality of catching memory leaks too. It is off by default - because it is not free for your application. Enabling this functionality means a little slow down and increased memory usage 589. This feature has its limits 589, but it can be very useful for debugging memory leaks on client's machines.

Note: memory leaks catching feature in EurekaLog is made as light-weight - to minimize performance/resource impact on your application. Thus, you can use it in your application deployed on end-user machines. However this means that EurekaLog provides less information than debugging solutions. The primary target of EurekaLog is to let you know about the problems. Surely, you can use EurekaLog to debug problems too, but it can be not suffice in some cases. So you may need to use other debugging tool which is designed to debug problems, not to report them "from the fields".

In any case, to enable this feature - you need to check "Catch memory leaks" option on "Memory problems" tab 250:



☑ Enable extended memory manager

Note: C++ Builder may require special setup for project. Please, see help for more info.

☑ Catch memory problems

☑ Fill freed memory with zeros

☐ Share memory manager

☐ Use existing shared memory manager

☑ Catch memory leaks

☑ Active only when running under debugger

☐ Allow manual activation control

☑ Group child memory leaks with its parent

☐ RAW stack tracing

☑ Hide RTL/VCL leaks

Minimal memory leak size to report (bytes): 0

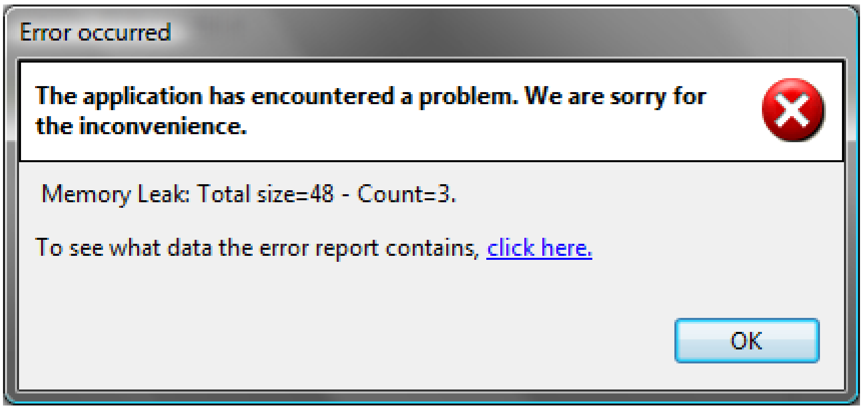
Maximum leak number to report: 0

(See the feature limits)

Memory debugging options

There are bunch of options which controls memory leaks checks activation and its behaviour. See this section 250 for more details about each option.

Anyway, if you activate memory leaks checks and there will be a memory leak in your application at run-time - there will be a usual error dialog at application's exit:



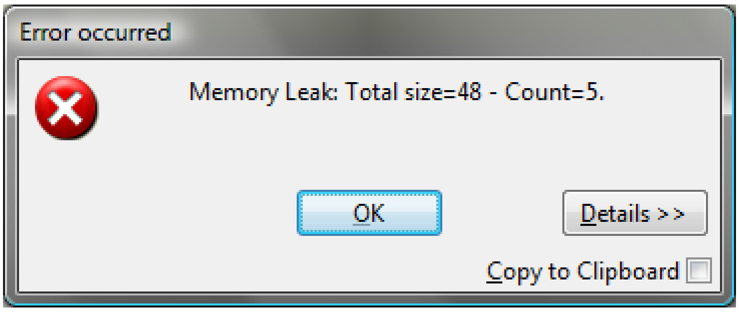
Error occurred

The application has encountered a problem. We are sorry for the inconvenience.

Memory Leak: Total size=48 - Count=3.

To see what data the error report contains, click here.

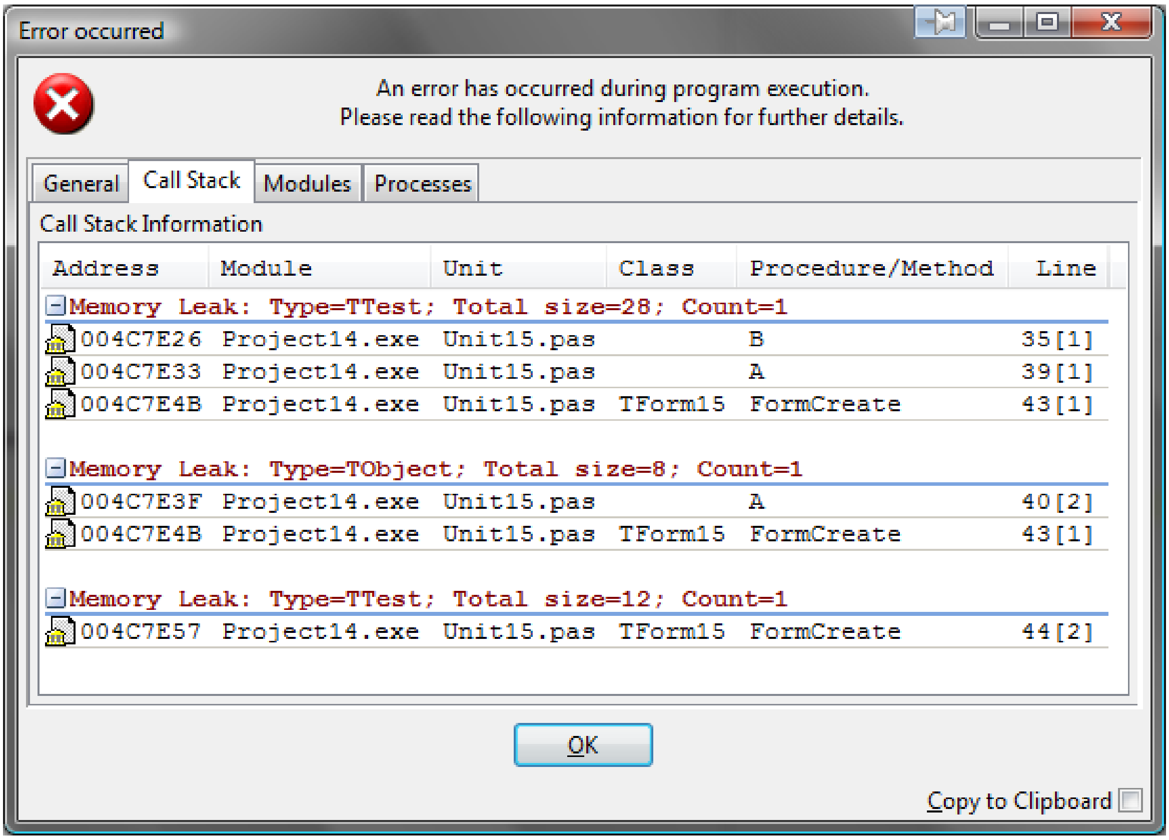
Memory leaks in MS Classic style dialog



Error occurred

Memory Leak: Total size=48 - Count=5.

Memory leaks in EurekaLog style dialog



Error occurred

An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

```
Address   Module        Unit        Class    Procedure/Method   Line
⊟Memory Leak: Type=TTest; Total size=28; Count=1
004C7E26  Project14.exe Unit15.pas           B                  35[1]
004C7E33  Project14.exe Unit15.pas           A                  39[1]
004C7E4B  Project14.exe Unit15.pas  TForm15  FormCreate         43[1]

⊟Memory Leak: Type=TObject; Total size=8; Count=1
004C7E3F  Project14.exe Unit15.pas           A                  40[2]
004C7E4B  Project14.exe Unit15.pas  TForm15  FormCreate         43[1]

⊟Memory Leak: Type=TTest; Total size=12; Count=1
004C7E57  Project14.exe Unit15.pas  TForm15  FormCreate         44[2]
```

Memory leaks in detailed dialog

As you can see: all memory leaks will be gathered in one single report, which can be send to you as any other EurekaLog report. The only differences from other kinds of reports are: no CPU and Assembler tabs and no calling of event handlers.

How to resolve memory leaks

Many people seems to miss the whole point of mem-leak reports. Typical approach of working with bug report for many people is to open code location from call stack and analyzing it. The problem is: **memory leak report does not point to the problem**.

Note: be sure to read Reading and understanding bug report 72 first, especially Searching bug's location 97 part.

Let's think for a second: what is a leak? Leak is... well, it is when we allocate something and do not release/free it. So, mem-leak report can (and, actually, will) contain that "something" - a resource; and it contains "allocation" - i.e. call stack to line of code, which allocates resource. But where is our problem? An actual problem is sitting at "release/free" moment! A tool can not know: where did you (your code) planned to release resource. That's why report contain only information about allocation. There is no direct information on the problem in the report.

What does it mean, "the problem is in release"? It means, that either we **lost pointer to resource** or we do have a pointer, but our **release routine wasn't called** for some reason. And those are points, which you should look at.

So, what should you do with mem-leak report? Well, you first need to follow call stack and find code. But the next thing is different (comparing to exception bug report): you don't need to analyze **this** line. You need:
1. Note, what resource was allocated here (object, string, array, memory block, etc...).
2. Find, where this resource should be released "by plan" (call do destructor, out of scope, explicit free call, etc...).
3. Found reason, why resource wasn't release at founded location.

As it was already said: there can be 2 reasons for item 3 - either we lost reference or we missed the call.

Delphi's bugs

Before starting doing anything - make sure, that the problem really exists: run your application in wild run without debugger. This will eliminate any possible false mem-leaks like this.

Aside from IDE's bugs, there can be bugs in RTL/VCL too: example. It can be direct bugs (and there is change for their fix in next Delphi version - example), or things that just can't be fixed. Anyway, both cases introduce a mem-leaks in your application and your code has nothing to do with it.

So what can you do here? Putting patching apart - the only thing you can do is to ignore them (since you can't fix them). Yes, this is a workaround. You don't fix a problem - you just hide it, so you can concentrate on problems, which you can fix. The main danger here is overuse of such routines: do NOT add all mem-leaks as "registered" - don't forget that this will not fix the problem!

Best Practices

Certain coding and design practices can limit the number of leaks in your code:
- Use managed data types with reference counting and smart pointers wrappers for non-managed types and functions (you will need to write your own wrapper classes).
- Be aware of leak patterns with managed types: circular references between objects.
- Avoid using multiple exit paths from a function. Allocations assigned to variables at function scope should be freed in one particular block at the end of the function.
- Use try/finally blocks to ensure (guarantee) finalization and dispose of memory and resources.
- Be careful with type-less functions in RTL. Any code which works with untyped argument must be carefully analyzed for leaks possibility.

See also:
- Configuring project for leaks detection 508
- Other memory problems 171
- Memory leaks settings 250
- Resource leaks settings 255
- EurekaLog memory leaks detection limitations 589
- EurekaLog resource leaks detection limitations 589

11.5.2 Configurations

11.5.2.1 Typical application

This section describes leaks configuration for usual application. It can be VCL Forms application, FMX application, console application, etc. **It should not use run-time packages; it should not use shared memory manager.**

Configuration is very simple, as there is no additional issues. You can enable/disable leaks control in EurekaLog options (see here: memory leaks 250 and resource leaks 255) - and that's it.

Detailed explanation

EurekaLog has concepts of "memory" and "resource" leaks:

- "Memory leaks" - are leaks from Delphi memory manager. Those leaks appear when code uses GetMem/FreeMem or any wrapper for it (such as AllocMem, TObject.Create, strings, dynamic arrays, etc.).
- "Resource leaks" - are leaks from any other allocation function (such as VirtualAlloc, HeapAlloc, etc. - see here 590 for detailed list).

EurekaLog has EMemLeaks and EResLeaks units. EMemLeaks unit contains code to catch "memory leaks", EResLeaks unit contains code to catch "resource leaks". These units will be added to your project's uses clause (use "Project" / "View source" IDE's menu command to view it) when you enable EurekaLog for your project.

Note: EMemLeaks and EResLeaks units are always included in your EurekaLog-enabled application - even if no leaks detection is turned on in your project's options. This is not a bug. Leaks detection can be turned on at run-time (via command-line switch or just programmatically by some code), and leaks detection code should be executed first (before any other code) - that's why EMemLeaks and EResLeaks units are always included.

Using leaks detection without EurekaLog

It is possible to use leaks detection without enabling entire EurekaLog for your project - just disable EurekaLog for your project and then enable corresponding leaks detection feature (or some other memory debugging features).

See also:

- Using leaks detection in DLLs 513
- Using leaks detection in applications with shared memory manager 513
- Using leaks detection in packaged applications 514

11.5.2.2 DLLs

DLLs are no different from applications in respect to leaks detection aspect. I.e. configuring typical DLL for leaks would be the same as configuring typical application for leaks; configuring packaged DLL for leaks would be the same as configuring packaged application for leaks; and so on. So:

- See Using leaks detection in typical applications 512 for learning more about configuring typical DLLs;
- See Using leaks detection in applications with shared memory manager 513 for configuring DLLs with shared memory manager;
- See Using leaks detection in packaged applications 514 for configuring DLLs with run-time packages.

DLL can use any ("standalone DLL 480" or "DLL 474") profile. This does not affect leaks detection configuration.

See also:

- Using leaks detection in typical applications 512
- Using leaks detection in applications with shared memory manager 513
- Using leaks detection in packaged applications 514

11.5.2.3 Shared memory manager

Enter topic text here.

11.5.2.4 Packaged project

Enter topic text here.

11.6 Using EurekaLog with other software

EurekaLog contains various configuration options 259 to allow you to integrate EurekaLog with other 3rd party software tools - such as external debuggers, profilers, executable packers, cryptors, protectors, debug information convertation tools, digital signing tools, etc.

Common information

Different tools have different requirements. Some tools require debug information, other tools require no changing in code (no hooks), etc. You can customize EurekaLog options for various tools on special page in options 259. Of course, disabling options will result in some sort of compromise: you will disable some features of EurekaLog, but gain more compatibility. Sometimes you will need to write some code to bring missed features back.

Generally, your usual method when integrating EurekaLog with other software should be as follows:

1. Configure your project for maximum possible debugging 58.
2. Try to compile your application with EurekaLog and external tool. You can use build events 351 to invoke external tool automatically when building the project.
3. If this doesn't work - open EurekaLog project options and go to "External tools 259" page.
4. Turn off one single option and repeat testing.
5. If this doesn't work - turn off more options.
6. Repeat this process until you get working solution.

The common rule of thumb is **not to blindly disable each possible EurekaLog options**. Each EurekaLog option allows you to use certain feature. If you disable each option without checking if this was needed at all - you may encounter unexpected behavior (like missed feature behavior). This is especially true for such options as "Handle every SafeCall exception", "Catch handled exceptions", memory manager options.

You can study hints for options (checkboxes) to read about recommended setup for your tool kind. You can start with switching off not recommended options first. You can also read detailed description of each option here 259.

Important note: it's highly recommended to keep "**Enable extended memory manager**" option turned on (you can disable other memory checking options if you want to). Installing filter on memory manager will allow EurekaLog to track life-time of exceptions objects without need to install code hooks.

There are some practical guides available:

- Debuggers/profilers 515
- Debug information converters 516
- Digital signing tools 519
- Packers/cryptors/protectors 520
- Localization tools 524
- 3rd party shared memory manager 524
- COM applications 488
- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
- Using EurekaLog with non-Embarcadero compilers (Microsoft Visual Studio, etc.) 496

A common post-processing order 351 is:

1. Debug information converters;
2. EurekaLog post-processing 421 (IDE expert or a call to ecc32/emake);
3. Compressors/protectors;
4. Digital signature tools;

See also:

- Options for integration with external software 259

- Troubleshooting EurekaLog work 613
- Internal errors 591
- Enabling debug information for system DLLs 504
- Configuring project for leaks detection 508

11.6.1 Debuggers and profilers

External debuggers and profilers requires source of debug information. You can keep EurekaLog hooks installed. Therefore, your actions to integrate with external debuggers or profiler would be:

1. Configure your project for maximum debugging 58. Be sure that required debug format option is turned on.
2. **Important note:** Disable "Delete service files after compilation" option 259 to keep debug information files around after compilation (so they can be used by debugger/profiler).
3. **Important note:** You may want to use `--el_DisableDebuggerPresent` command-line switch to disable communication between EurekaLog and debugger (such as naming threads and output debug information).
4. You may need help from additional tools 516.

Example: AQTime

1. Create or open the project.
2. Enable and configure EurekaLog.
3. Open project options 58:
   a. Set "Linking / Map file" = "Detailed" (Delphi) or "Detailed segment map" (C++ Builder). Be sure that "Map with mangled names" option is turned off (if it's present).
   b. Enable "Linking / Debug information" (new Delphi), "Include TD32 debug info" (old Delphi) or "Full debug information" (C++ Builder). Be sure that "Place debug information in separate TDS file" option is turned on (if it's present).
   c. Enable "Stack frames", "Debug information", "Use Debug DCUs" and (optionally) "Range checking" options on "Compiling" page (Delphi only).
   d. Enable "Debug information" and "Debug line number information" options on "C++ Compiler"/"Debugging" page (C++ Builder only).
4. Open EurekaLog project options:
   a. Disable "Delete service files after compilation" option 259.
5. (Optional) Add `--el_DisableDebuggerPresent` command-line switch to parameters of your application. Note: EurekaLog could auto-detect AQTime, so adding `--el_DisableDebuggerPresent` command-line switch is not strictly necessary for AQTime, but it may be required for other 3rd party tools.
6. (Optional) You may use `/EL_DisableMemoryFilter` or `/EL_DisableMemoryLeaks` command-line options 250.
7. Run your application with AQTime.
8. AQTime *may* display a warning about some EurekaLog routines could not be profiled due to small size. You can safely ignore this warning.

If you've done everything correctly - you should see a profiling report after closing application. If you didn't setup your project or didn't disable "Delete service files after compilation" option - then you should see blank report. If you didn't specified `--el_DisableDebuggerPresent` command-line switch - then your application may crash at start time, because EurekaLog would detect the IDE debugger and try to communicate with it.

Note: EurekaLog files are compiled with debug information by default. This means that you'll see EurekaLog routines in final report. You may want to disable memory/resource leaks checking to reduce noise. You can setup AQTime filtering (please refer to AQTime help system). Or you may recompile EurekaLog 619 with disabled debug information.

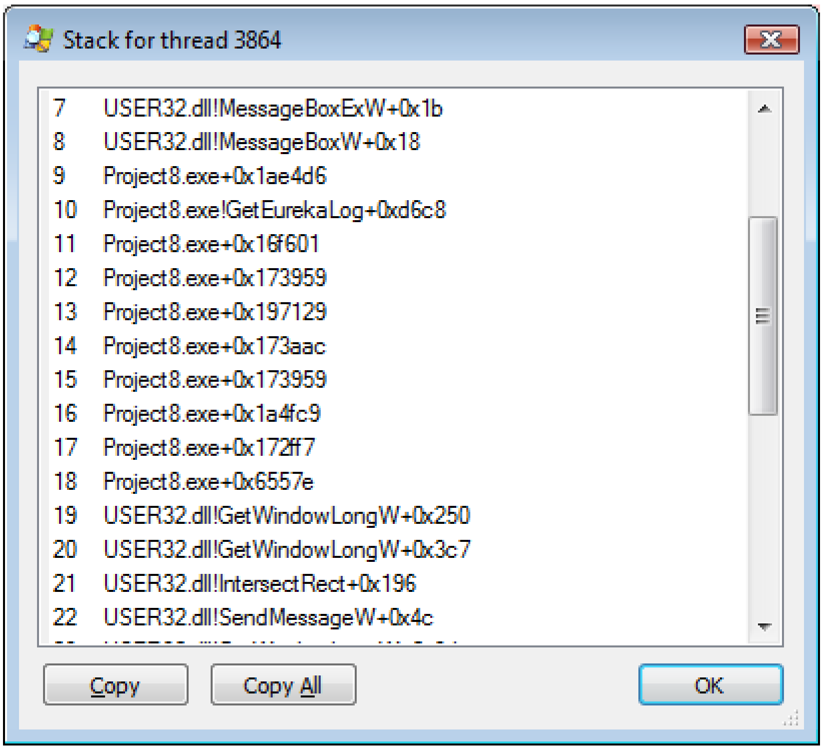
See also:

- Using EurekaLog with other software 514
- Troubleshooting EurekaLog work 613

11.6.2 Debug information converters

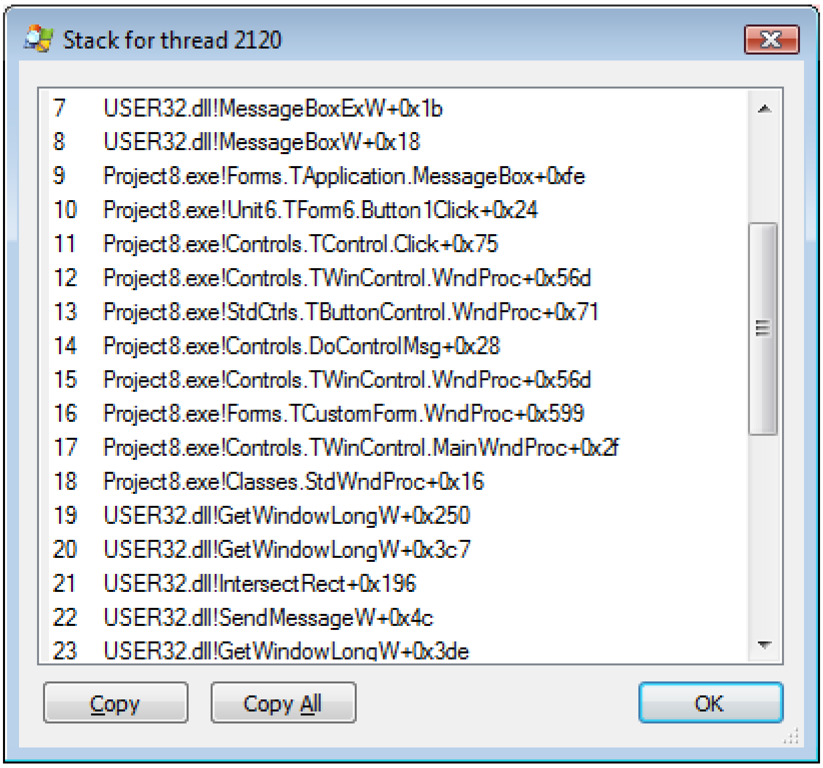
Debug information is necessary to display human-readable code location descriptions. It's used by many debugging-related tools to show readable information instead of RAW pointers. There are different formats of debug information. Each tool understands some of these formats. Another tool may understand other formats. If particular tool doesn't understand debug information format of your executable - then it will not show any useful information. For example, Delphi and C++ Builder supports variety of debug formats: embedded dcu/obj, .map (output only), .tds (TD32), .rsm (remote debug), DWARF (currently: 64-bit C++ Builder only) - all of these (except for DWARF) are Borland/CodeGear/Embarcadero-centric. On the other hand, Microsoft tools support Microsoft formats: .dbg and .pdb. Obviously, Embarcadero and Microsoft tools do not understand each other.

You can use debug information converters to solve such issues. These converter tools will take some form of debug information as input and convert it into another form.



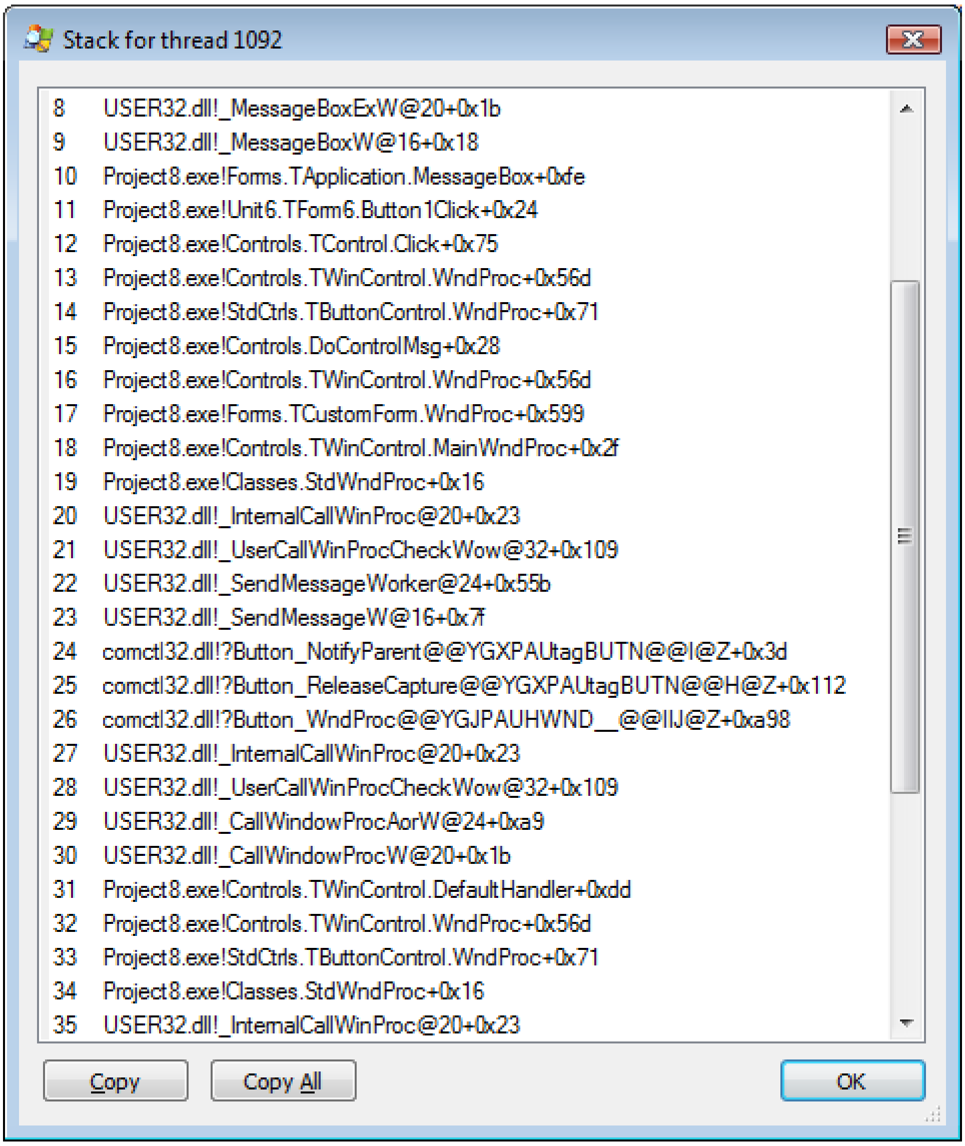
```
7   USER32.dll!MessageBoxExW+0x1b
8   USER32.dll!MessageBoxW+0x18
9   Project8.exe+0x1ae4d6
10  Project8.exe!GetEurekaLog+0xd6c8
11  Project8.exe+0x16f601
12  Project8.exe+0x173959
13  Project8.exe+0x197129
14  Project8.exe+0x173aac
15  Project8.exe+0x173959
16  Project8.exe+0x1a4fc9
17  Project8.exe+0x172ff7
18  Project8.exe+0x6557e
19  USER32.dll!GetWindowLongW+0x250
20  USER32.dll!GetWindowLongW+0x3c7
21  USER32.dll!IntersectRect+0x196
22  USER32.dll!SendMessageW+0x4c
```

Process Explorer tool shows no useful information about Delphi project



```
7    USER32.dll!MessageBoxExW+0x1b
8    USER32.dll!MessageBoxW+0x18
9    Project8.exe!Forms.TApplication.MessageBox+0xfe
10   Project8.exe!Unit6.TForm6.Button1Click+0x24
11   Project8.exe!Controls.TControl.Click+0x75
12   Project8.exe!Controls.TWinControl.WndProc+0x56d
13   Project8.exe!StdCtrls.TButtonControl.WndProc+0x71
14   Project8.exe!Controls.DoControlMsg+0x28
15   Project8.exe!Controls.TWinControl.WndProc+0x56d
16   Project8.exe!Forms.TCustomForm.WndProc+0x599
17   Project8.exe!Controls.TWinControl.MainWndProc+0x2f
18   Project8.exe!Classes.StdWndProc+0x16
19   USER32.dll!GetWindowLongW+0x250
20   USER32.dll!GetWindowLongW+0x3c7
21   USER32.dll!IntersectRect+0x196
22   USER32.dll!SendMessageW+0x4c
23   USER32.dll!GetWindowLongW+0x3de
```

The same project after its debug information converted to DBG format



```
8     USER32.dll!_MessageBoxExW@20+0x1b
9     USER32.dll!_MessageBoxW@16+0x18
10    Project8.exe!Forms.TApplication.MessageBox+0xfe
11    Project8.exe!Unit6.TForm6.Button1Click+0x24
12    Project8.exe!Controls.TControl.Click+0x75
13    Project8.exe!Controls.TWinControl.WndProc+0x56d
14    Project8.exe!StdCtrls.TButtonControl.WndProc+0x71
15    Project8.exe!Controls.DoControlMsg+0x28
16    Project8.exe!Controls.TWinControl.WndProc+0x56d
17    Project8.exe!Forms.TCustomForm.WndProc+0x599
18    Project8.exe!Controls.TWinControl.MainWndProc+0x2f
19    Project8.exe!Classes.StdWndProc+0x16
20    USER32.dll!_InternalCallWinProc@20+0x23
21    USER32.dll!_UserCallWinProcCheckWow@32+0x109
22    USER32.dll!_SendMessageWorker@24+0x55b
23    USER32.dll!_SendMessageW@16+0x7f
24    comctl32.dll!?Button_NotifyParent@@YGXPAUtagBUTN@@I@Z+0x3d
25    comctl32.dll!?Button_ReleaseCapture@@YGXPAUtagBUTN@@H@Z+0x112
26    comctl32.dll!?Button_WndProc@@YGJPAUHWND__@@IIJ@Z+0xa98
27    USER32.dll!_InternalCallWinProc@20+0x23
28    USER32.dll!_UserCallWinProcCheckWow@32+0x109
29    USER32.dll!_CallWindowProcAorW@24+0xa9
30    USER32.dll!_CallWindowProcW@20+0x1b
31    Project8.exe!Controls.TWinControl.DefaultHandler+0xdd
32    Project8.exe!Controls.TWinControl.WndProc+0x56d
33    Project8.exe!StdCtrls.TButtonControl.WndProc+0x71
34    Project8.exe!Classes.StdWndProc+0x16
35    USER32.dll!_InternalCallWinProc@20+0x23
```

Same project after enabling Microsoft debug information for system DLLs

There are tools (like map2dbg or tds2pdb) which are able to convert map/tds to dbg/pdb - to make integration with Microsoft tools possible (such as Process Explorer or WinDbg). Basically, such converter tools require executable and debug information source. You can call them after compilation of your project to convert debug information file. Therefore, your actions to integrate with debug information converter would be:

1. Configure your project for maximum debugging 58. Be sure that required debug format option is turned on.
2. Disable "Delete service files after compilation" option 259 to keep debug information files around after compilation (so they can be used by debugger/profiler).
3. Place a call to converter tool to post-build event 351.

Example: Process Explorer and map2dbg tool

1. Create or open the project.
2. Enable and configure EurekaLog.
3. Open project options 58:
   a. Set "Linking / Map file" = "Detailed" (Delphi) or "Detailed segment map" (C++ Builder). Be sure that "Map with mangled names" option is turned off (if it's present).
   b. Enable "Linking / Debug information" (new Delphi), "Include TD32 debug info" (old Delphi) or "Full debug information" (C++ Builder). Be sure that "Place debug information in separate TDS file" option is turned on (if it's present).

   c. Enable "Stack frames", "Debug information", "Use Debug DCUs" and (optionally) "Range checking" options on "Compiling" page (Delphi only).
   d. Enable "Debug information" and "Debug line number information" options on "C++ Compiler"/"Debugging" page (C++ Builder only).

4. Open EurekaLog project options:
   a. Disable "Delete service files after compilation" option 259.
5. Open build events 351 page and place a call to map2dbg tool to post-build event (success): `map2dbg "%_IDETarget%"`
6. Build your project.

Notes:

- Do not confuse post-build's *success* and *failure* events. You need to insert call to post-build **success** event, not into post-build failure event.
- You may need to specify full file path for tool's .exe file (like `C:\Tools\map2dbg.exe`).
- Do not forget about surrounding double quotes for files with spaces in path.
- Delphi/C++ Builder 2007+ also have build events. You can use either EurekaLog's or IDE's build events.
- Debug information converter tool may run before or after EurekaLog's post-processing 426.

If you've done everything correctly - you should see a information about your Delphi/C++ Builder project in Process Explorer tool. Run your application, launch Process Explorer, right-click on your project in Process Explorer and click on "Properties", switch to "Threads" tab, pick a thread and click on "Call stack" button. You should see readable locations about your code instead of just module name + RAW offset.

See also:

- Configuring Microsoft symbols 504
- Using EurekaLog with other software 514
- Post-processing without (re)compilation 426
- Tools:
  - map2dbg
  - tds2pdb
  - tds2pdb (another author)
  - tdstrp32
  - cv2pdb

11.6.3 Digital signature tools

Digital signing tools (such as SignTool.exe, X2NetSignCode.exe, etc.) can be used to digitally sing your executable. Digital signing process is also known as code signing. You will need some code signing tool, digital certificate and a internet connection on the build machine.

Digital signature allows software developers to include information about themselves and their code with their software. It also prevent changes in executables:

- Content Source: End users can confirm that the software really comes from the publisher who signed it.
- Content Integrity: End users can verify that the software has not been altered or corrupted since it was signed.

Normally, digital signing tools do not have any special requirements on executable. Of course, you still need certificate and internet connection (for time-stamping), but executable can be anything. No debug information is required either. Hooks and run-time modifications are allowed without restrictions. The only possible issue: digital signing may conflict with EurekaLog's own integrity check (CRC calculations). Therefore, your actions to integrate with digital signing tools would be:

1. Disable "Check file for corruption" option 259 to disable EurekaLog's checks (as they will be superseded by digital signature).
2. Place a call to digital signing tool post-build event 351.

Example: automatically digitally sign executable on build

1. Create or open the project.

2. Enable and configure EurekaLog.
3. Open EurekaLog project options:
   a. Disable "Check file for corruption" option 259.
4. Open build events 351 page and place a call to digital signing tool to post-build event (success). Exact command line depends on your tool. Usually there should be argument to indicate certificate file(s), time-stamping service, password for private key, description of executable, and executable itself. You can use `%_IDETarget%` pseudo-variable to automatically point to final executable file.
5. Build your project.

Notes:

- Do not confuse post-build's *success* and *failure* events. You need to insert call to post-build **success** event, not into post-build failure event.
- You may need to specify full file path for tool's .exe file (like `C:\Tools\signtool.exe`).
- Do not forget about surrounding double quotes for files with spaces in path.
- Delphi/C++ Builder 2007+ also have build events. **Do not place call to signing tool to IDE's build events**. Build events of IDE are executed as part of compilation - thus, IDE's post-build event will fire before EurekaLog will be able to do its post-processing. This means that EurekaLog's post-processing will modify executable and break digital signature. Therefore, you should use EurekaLog's post-build events to call signing tool only after compilation and post-processing.
- It's not strictly necessary to disable "Check file for corruption" option. This option uses standard CRC checksum field in PE header. Normally digital signing process will update this field to reflect changes. However, even though this option still may work after signing - it may be not needed, because this work will be done by OS loader when checking digital signature on load. On the other hand, default configuration of OS allows running executable with broken digital signature, so you may want to keep "Check file for corruption" option turned on.

See also:

- Using EurekaLog with other software 514

11.6.4 EXE packers, EXE cryptors, EXE protectors

Executable compression is any means of compressing an executable file and combining the compressed data with decompression code into a single executable. When this compressed executable is executed, the decompression code recreates the original code from the compressed code before executing it. In most cases this happens transparently so the compressed executable can be used in exactly the same way as the original. Examples of executable packer tools: UPX, ASPack, Smart Pack Pro, etc.

Executable cryptor and protector tools transforms executable file in order to protect it against disassembling and reverse engineering. This usually involves many different techniques: encryption, packing, VM code, using hook/redirections, detecting debuggers, etc. Examples of protector tools: ASProtect, Themida, VMProtect, etc.

Using packers and protectors tools with exception tracer tool (such as EurekaLog) is a tricky task. That's because using exception tracer basically means "intercept exceptions", and protectors means "do not allow to mess with this process". Therefore, exception tracer wants to install hook (modify running process), protectors wants to prevent hooks. Clearly these two goals are in direct conflict with each other.

You may encounter the following issues:

- (EXE file changes) Packer/protector tool may detect injecting EurekaLog's data during post-processing. The solution is to inject EurekaLog's data first and then post-process executable with packer/protector.
- (Running process changes) Packer/protector tool may detect code hooks installed by EurekaLog. The solution is to avoid installing hooks when possible.
- (Breaking code <-> debug info corresponding) Packer/protector tool may alter executable code in such way so it will be incompatible with debug information. There is no solution to this issue. However, most software is trying to preserve original code after unpacking.
- (Conflicting tricks) Protector tools may use different kinds of tricks for protection. Some of these techniques may conflict with exception tracer. Such issues can usually be solved by adjusting options.

You can disable the following options in EurekaLog:
- Check file corruption 259
- Use speed optimizations 259
- Use low-level hooks 259
- Handle every SafeCall exception 259
- Catch handled exceptions 259
- Code hooks 352
- Debugging memory manager 250

Important note: it's highly recommended to keep "**Enable extended memory manager**" option turned on (you can disable other memory checking options if you want to). Installing filter on memory manager will allow EurekaLog to track life-time of exceptions objects without need to install code hooks.

See also: Using EurekaLog and 3rd party shared memory manager 524.

Alternatively, you can also try to use the following options in your packer/protector tool:
- Do not touch resources (if possible)
- Avoid using VM (if possible)
- Protect file instead of running process (if possible)
- Disable debugger/hooks detection (if possible)
- Disable virtualization of OS functions (if possible)

Generally, if your packer/protector tool has some options to integrate with digital signatures - it would be a good idea to enable these options for using with EurekaLog too.

General information

Many protector tools allow you to wrap blocks of code into some sort of markers to indicate that such code block should be protected/encrypted/virtualized (by replacing it with VM code). Some code will be dynamically generated. Such modifications will break binary code <-> source code interoperability (via debug information). Therefore, EurekaLog will not be able to show meaningful information about such code. **Heavy usage of VM/dynamical code in your application will significantly weaken features of EurekaLog**. Use only a minimally possible amount of such code.

There are difference in using IDE and environments. **The best option is to use Windows XP or later, Delphi 2009 or later, and Win64 platform.**
- Windows XP allows EurekaLog-enabled application to use VEH (Vectored Exception Handling) to capture CPU state for exceptions. Older systems require using low-level hooks for that task (which may have problem with protectors and anti-virus software).
- Delphi 2009 and above has extended exceptions support. A better integration with exception tracer tool is possible. EurekaLog can integrate with RTL without using any hooks on Delphi 2009+.
- Win64 platform is different from Win32 platform because there is no reliable method to build a call stack on Win32 (so each method uses heuristic to guess possible code locations), but each function on Win64 must contain a stack frame (i.e. declare itself). Therefore, stack tracing is always accurate on Win64 even if dynamic code is used.

"Check file corruption" option should be turned off, as almost each packer/protector already contain the same functionality. However, it's not strictly necessary. This option uses standard CRC checksum field in PE header. Normally packer and protector tools will update this field to reflect changes. However, even though this option still may work after packing/protecting - it may be not needed, because this work will be done by packer/protector integrity checks. Unlike digital signatures, packer and protector will refuse to run modified executable.

"Use low-level hooks" and "Code hooks" options installs hooks on routines in code. This may cause problems with integrity checks in protectors. **Disable these options to avoid it**. Low-level hooks can be replaced by using Windows XP+ and Delphi 2009+. Code hooks may be replaced with manual invoke of exception manager 370. See also practical examples below.

"Use speed optimizations" options caches kernel information to speed up stack tracing process. This feature may be in conflict with different protector's tricks. **Disable this options to avoid it**.

The common rule of thumb is **not to blindly disable each possible EurekaLog options**. Each EurekaLog option allows you to use certain feature. If you disable each option without checking if this was needed at all - you may encounter unexpected behavior (like missed feature behavior). This is especially true for such options as "Handle every SafeCall exception", "Catch handled exceptions", memory manager options. Therefore the best way is to disable options one-by-one and checking work status on each step:

1. Configure your project for maximum possible debugging 58.
2. Try to compile your application with EurekaLog and packer/protector. You can use build events 351 to invoke external tool automatically when building the project.
3. If this doesn't work - open EurekaLog project options and go to "External tools 259" page.
4. Turn off one single option and repeat testing.
5. If this doesn't work - turn off more options.
6. Repeat this process until you get working solution.

Example: using EurekaLog with UPX

1. Create or open the project.
2. Enable and configure EurekaLog.
3. Open build events 351 page and place a call to UPX to post-build event (success): `UPX.exe -9 "%_IDETarget%"`
4. Build your project.

Notes:

- Do not confuse post-build's *success* and *failure* events. You need to insert call to post-build **success** event, not into post-build failure event.
- You may need to specify full file path for tool's .exe file (like `C:\Tools\upx.exe`).
- Do not forget about surrounding double quotes for files with spaces in path.
- Delphi/C++ Builder 2007+ also have build events. **Do not place call to UPX to IDE's build events**. Build events of IDE are executed as part of compilation - thus, IDE's post-build event will fire before EurekaLog will be able to do its post-processing. This means that EurekaLog's post-processing will try to modify already packed executable. Therefore, you should use EurekaLog's post-build events to call packer only after compilation and post-processing.
- There are no special requirements for options, but you may also want to try to disable the above mentioned options (if you've encountered any issue). "Check file corruption" option is a good candidate to be turned off (even if it's not strictly necessary).
- You may want to use UPX's options like `--compress-resources=0` or `--keep-resources=`*`list`* to exclude EurekaLog's data from being compressed (EurekaLog's data is already compressed).
- You may try to use UPX's option `--compress-exports=0` to avoid changing exports table.

Example: using EurekaLog with ASProtect

ASProtect tool is awared of Delphi/C++ Builder IDE and will ask you for .map file. Therefore, it must be enabled in options 58 and "Delete service files after compilation" option 259 must be turned off.

1. Create or open the project.
2. Enable and configure EurekaLog.
3. Open project options 58:
   a. Set "Linking / Map file" = "Detailed" (Delphi) or "Detailed segment map" (C++ Builder). Be sure that "Map with mangled names" option is turned off (if it's present).
   b. Enable "Linking / Debug information" (new Delphi), "Include TD32 debug info" (old Delphi) or "Full debug information" (C++ Builder). Be sure that "Place debug information in separate TDS file" option is turned on (if it's present).
   c. Enable "Stack frames", "Debug information", "Use Debug DCUs" and (optionally) "Range checking" options on "Compiling" page (Delphi only).
   d. Enable "Debug information" and "Debug line number information" options on "C++ Compiler"/"Debugging" page (C++ Builder only).
4. Open EurekaLog project options:

   a. Disable "Delete service files after compilation" option 259.
   b. Disable "Check file corruption" option 259.
   c. Disable "Use low-level hooks" option 259.
   d. Disable all "Code Hooks" options 352. **Note:** your application type will be converted to "Custom settings / Unsupported type". This is normal/expected behavior.
   e. Enable extended memory manager 250. You can enable only extended memory manager without any additional options.

5. Add `EExceptionManager` unit to uses clause of your main form's unit.
6. Place `TApplicationEvents` component on your main form.
7. Create new `OnException` event handler for `TApplicationEvents` component.
8. Place a "`ExceptionManager.Handle(E)`" call inside event handler for `OnException` event.
9. Build your application and check that it's working as you expect it to work.
10. Build your project with ASProtect. You can use post-build event 351 to automatically invoke ASProtect on each compilation: `ASProtect.exe -process Project1.aspr2`

Notes:

- Do not confuse post-build's *success* and *failure* events. You need to insert call to post-build **success** event, not into post-build failure event.
- You may need to specify full file path for tool's .exe file (like `C:\Tools\ASProtect.exe`).
- Do not forget about surrounding double quotes for files with spaces in path.
- Delphi/C++ Builder 2007+ also have build events. **Do not place call to ASProtect to IDE's build events**. Build events of IDE are executed as part of compilation - thus, IDE's post-build event will fire before EurekaLog will be able to do its post-processing. This means that EurekaLog's post-processing will modify already protected executable. Therefore, you should use EurekaLog's post-build events to call protector tool only after compilation and post-processing.

Example: using EurekaLog with Themida

1. Create or open the project.
2. Enable and configure EurekaLog.
3. Open project options 58:
   a. Set "Linking / Map file" = "Detailed" (Delphi) or "Detailed segment map" (C++ Builder). Be sure that "Map with mangled names" option is turned off (if it's present).
   b. Enable "Linking / Debug information" (new Delphi), "Include TD32 debug info" (old Delphi) or "Full debug information" (C++ Builder). Be sure that "Place debug information in separate TDS file" option is turned on (if it's present).
   c. Enable "Stack frames", "Debug information", "Use Debug DCUs" and (optionally) "Range checking" options on "Compiling" page (Delphi only).
   d. Enable "Debug information" and "Debug line number information" options on "C++ Compiler"/"Debugging" page (C++ Builder only).
4. Open EurekaLog project options:
   a. Disable "Check file corruption" option 259.
   b. Disable "Use low-level hooks" option 259.
   c. Disable all "Code Hooks" options 352. **Note:** your application type will be converted to "Custom settings / Unsupported type". This is normal/expected behavior.
   d. Enable extended memory manager 250. You can enable only extended memory manager without any additional options.
5. Add `EExceptionManager` unit to uses clause of your main form's unit.
6. Place `TApplicationEvents` component on your main form.
7. Create new `OnException` event handler for `TApplicationEvents` component.
8. Place a "`ExceptionManager.Handle(E)`" call inside event handler for `OnException` event.
9. Build your application and check that it's working as you expect it to work.
10. Build your project with Themida. You can use post-build event 351 to automatically invoke Themida on each compilation: `Themida.exe /protect Project1.tmd`

Notes:

- Do not confuse post-build's *success* and *failure* events. You need to insert call to post-build **success** event, not into post-build failure event.
- You may need to specify full file path for tool's .exe file (like `C:\Tools\Themida.exe`).
- Do not forget about surrounding double quotes for files with spaces in path.
- Delphi/C++ Builder 2007+ also have build events. **Do not place call to Themida to IDE's build events**. Build events of IDE are executed as part of compilation - thus, IDE's post-build event will fire before EurekaLog will be able to do its post-processing. This means

that EurekaLog's post-processing will modify already protected executable. Therefore, you should use EurekaLog's post-build events to call protector tool only after compilation and post-processing.

See also:
- Using EurekaLog with other software 514
- Troubleshooting EurekaLog work 613

11.6.5 Localization software

External localization software may require debug information or other support information generated by compiler/linker. It usually does not interfere with EurekaLog in any way, so you can keep EurekaLog hooks installed. Therefore, your actions to integrate with external debuggers or profiler would be:
1. Configure your project for maximum debugging 58. Be sure that required debug format option is turned on.
2. Disable "Delete service files after compilation" option 259 to keep debug information files around after compilation (so they can be used by debugger/profiler).
3. You may need help from additional tools 516.

See also:
- Using EurekaLog with other software 514
- Troubleshooting EurekaLog work 613

11.6.6 Shared memory manager

Warning: this topic talks about using **shared** memory manager only. If you do not share memory manager between modules - then you can use any memory manager in any combinations without limitations.

EurekaLog is compatible with shared memory manager from FastMM. EurekaLog is not compatible with other implementations of shared memory manager. Therefore, if you're using shared memory manager - then you must select (and follow) one of the following scenario:
- all modules doesn't use EurekaLog (you use 3rd party shared memory manager)
- all modules use EurekaLog, but "Enable extended memory manager" option is turned off for all modules (you use 3rd party shared memory manager)
- all modules use EurekaLog and both "Enable extended memory manager" and "Share memory manager" (or "Use existing shared memory manager") option are turned on for all modules (you use EurekaLog's shared memory manager)
- (special: FastMM integration) some modules use FastMM with sharing enabled, some modules use EurekaLog with both "Enable extended memory manager" and "Share memory manager" (or "Use existing shared memory manager") option turned on for all EurekaLog-enabled modules (you use either EurekaLog's or FastMM's memory manager - see below for more info)

In other words, you can't have shared memory manager and different settings for "Enable extended memory manager" option in several modules. Consider enabling "Enable extended memory manager" option as using another memory manager. Obviously, in order to use shared memory manager, you must use the same memory manager in all modules. That's why you can't have different settings for "Enable extended memory manager" option.

If you're **not** using shared memory manager - then "Enable extended memory manager" option can be set differently for each individual module. Also, "Share memory manager" and "Use existing shared memory manager" options should be unchecked.

Using together EurekaLog and FastMM

If you're **not** using shared memory manager - you can mix FastMM and EurekaLog in your modules in any combination.

If you're using shared memory manager - you must enable "ShareMM" option for FastMM for non-EurekaLog modules (please, refer to FastMM's documentation) and you must enable both "Enable extended memory manager" and "Share memory manager" options for EurekaLog's modules. With these settings - you can mix FastMM and EurekaLog in your modules in any combination.

Warning: do not enable "ShareMM" option in FastMM for EurekaLog's modules (in case you use both FastMM and EurekaLog in the same module), unless you disable "Enable extended memory manager" option.

Since EurekaLog is compatible with FastMM - you can (for example) compile .exe with EurekaLog and both "Enable extended memory manager" and "Share memory manager" options checked; and you can compile DLL with FastMM and "ShareMM" option enabled (and without EurekaLog). This will work, and application will use EurekaLog's debugging features if .exe is initialized first. However, if you statically link DLL to .exe - DLL will be initialized first and FastMM will be used as application's memory manager. Thus, EurekaLog's memory debugging capabilities will be disabled.

To always use EurekaLog's memory manager:
- do not statically link to DLLs, use dynamic loading

or
- use EurekaLog in all modules

or
- do not use shared memory manager

Note: if you're using both EurekaLog and FastMM in the same project **-** be sure to list FastMM's unit first and EurekaLog's unit seconds in uses clause of main .dpr file - regardless of options set.

Using together EurekaLog and Delphi

Starting with Delphi 2007 - there are standard `AttemptToUseSharedMemoryManager` and `ShareMemoryManager` functions, which you can use for sharing memory manager. These functions are wrappers for FastMM's sharing functionality (which is a standard memory manager since Delphi 2006).

Thus, this case is the same as "Using together EurekaLog and FastMM", but instead of FastMM and setting "ShareMM" option you should use `ShareMemoryManager` function.

Warning: if you're using `ShareMemoryManager` function and EurekaLog in the same project - be sure to disable "Enable extended memory manager" option.

Note: if you're using both EurekaLog and `ShareMemoryManager` function in the same project **-** be sure to call `ShareMemoryManager` function before running EurekaLog's units. You can do this by including unit which calls `ShareMemoryManager` function in its initialization section. You must list this unit before any EurekaLog's units in uses clause of main .dpr file - regardless of options set.

Usually you use EurekaLog with both "Enable extended memory manager" and "Share memory manager" options checked for executable, and you use `ShareMemoryManager` function for dynamically loaded DLLs.

Using together EurekaLog and SimpleShareMem

SimpleShareMem is wrapper for `AttemptToUseSharedMemoryManager` and `ShareMemoryManager` functions (see above).

Thus, this case is the same as "Using together EurekaLog and FastMM", but instead of FastMM and "ShareMM" you should use SimpleShareMem unit.

Warning: if you're using SimpleShareMem and EurekaLog in the same project - be sure to disable "Enable extended memory manager" option.

Note: if you're using both EurekaLog and SimpleShareMem in the same project **-** be sure to list SimpleShareMem unit first and EurekaLog's unit seconds in uses clause of main .dpr file - regardless of options set.

Usually you use EurekaLog with both "Enable extended memory manager" and "Share memory manager" options checked for executable, and you use SimpleShareMem unit for dynamically loaded DLLs.

Using together EurekaLog and borlndmm.dll or ShareMem

EurekaLog is **not** compatible with BorlandMM.dll and ShareMem. It's recommended to use FastMM with "ShareMM" option instead. If this is not possible - then you must disable "Enable extended memory manager" option for all your EurekaLog-enabled modules. Using of EurekaLog's debugging capabilities will be not possible.

Note: other EurekaLog's features (not memory-related) still will be accessible.

Using together EurekaLog and dynamic RTL in C++ Builder

EurekaLog is **not** compatible with BorlandMM.dll and dynamic RTL in C++ Builder. It's recommended to use statically-linked RTL instead. If this is not possible - then you must disable "Enable extended memory manager" option for all your EurekaLog-enabled modules. Using of EurekaLog's debugging capabilities will be not possible.

Note: other EurekaLog's features (not memory-related) still will be accessible.

Using together EurekaLog and other 3rd party memory manager

EurekaLog is **not** compatible with other shared memory managers. It's recommended to use FastMM with "ShareMM" option instead. If this is not possible - then you must disable "Enable extended memory manager" option for all your EurekaLog-enabled modules. Using of EurekaLog's debugging capabilities will be not possible. You may only use capabilities of your shared memory manager, but not EurekaLog.

Note: other EurekaLog's features (not memory-related) still will be accessible.

What can go wrong

If you'll use wrong settings - you can get these issues:

- false-positive memory leaks;
- missing memory leaks;
- heap corruption reports;
- access violations;
- application's crash.

Typical reasons for these issues include the following:

- Your code lists units in wrong order. For example, FastMM/SimpleShareMem/etc is not the first unit in uses of .dpr;
- Your code uses conflicted settings. For example, DLL is compiled with ShareMem, but .exe is compiled with EurekaLog's "Enable extended memory manager" option.

See also:

- Memory problems options 250
- Memory leak detection limitations 589
- Enabling memory features for C++ Builder 255
- Using EurekaLog with other software 514

11.6.7 Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.)

EurekaLog supports reading of some 3rd party formats of debug information. This feature could be used in a migration scenario: when you migrate your multi-DLL project from other solution (such as JCL, madExcept, etc.) to EurekaLog. You can re-use old DLLs without recompiling these DLLs for EurekaLog. Your application should use the single exception tracer scheme 474.

1. Your host (.exe file) should be EurekaLog-enabled with enabled support for 4rd party debug information formats (see below).
2. Your DLL files could be:
   - o EurekaLog-enabled DLLs (using "DLL profile"[368])
     OR
   - o DLLs with 3rd party debug information which were post-processed by 3rd party compilers (such as JCL, madExcept, etc.)

You can mix EurekaLog-enabled DLLs and "3rd party-enabled DLLs" in the same application. In other words, EurekaLog-enabled DLLs with "DLL" profile (i.e. without exception tracer in DLL) are interchangeable with DLLs post-processed by 3rd party tools (without including a working exception tracer in DLL). "Standalone DLL" profile is not compatible with 3rd party exception tracers.

Host application should have ability to read debug information from DLLs. EurekaLog supports many formats of debug information[409]. Support for EurekaLog's own format of debug information is always enabled. Other formats should be enabled manually in EurekaLog project's options[355].

Note: there is no support to convert debug information from 3rd party post-processor tools into EurekaLog debug information format. That's because all such formats are very similar to each other. There is no significant benefit from converting debug information from some debug information format to another format. Therefore you should just enable support for particular format in your application. No convertation is necessary.

See also:
- Single instance of exception tracer[474]
- Debug information providers[409]
- Debug information providers configuration[355]

11.6.8 Using EurekaLog with non-Embarcadero DLLs

EurekaLog can be used with DLLs compiled by non-Embarcadero compilers - such as Microsoft Visual Studio, etc. 3rd party compiler must generate debug information in some of supported by EurekaLog formats (see list[409]). Your application should use the single exception tracer scheme[474].

Note: PDB format is a modern debug information format for Microsoft Visual Studio tool chain. It can contain much more information than older DBG debug information format. DBG format support is limited in many tools. For example. It is recommended to use PDB format when possible.

You can mix EurekaLog-enabled DLLs and 3rd party compilers DLLs in the same application. In other words, EurekaLog-enabled DLLs with "DLL" profile (i.e. without exception tracer in DLL) are interchangeable with DLLs compiled by 3rd party compiler for all of the 3 cases above (i.e. converting debug information to EurekaLog format, enabling support for additional formats, or using plain DLL exports information). "Standalone DLL" profile is not compatible with 3rd party exception tracers.

For the purposes of this article we will use the following sample code:

MSSample.cpp file:

```
#include "stdafx.h"
#include "MSSample.h"

void InternalTest(void)
{
   int * P;
   P = NULL;
   *P = 0;
}

void Test(void)
{
   InternalTest();
}

MSSAMPLE_API int fnMSSample(void)
{
   Test();
   return 42;
}
```

MSSample.h file:

```
#ifdef MSSAMPLE_EXPORTS
#define MSSAMPLE_API __declspec(dllexport)
#else
#define MSSAMPLE_API __declspec(dllimport)
#endif

MSSAMPLE_API int fnMSSample(void);
```

Unit1.pas file:

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TTestProc = function: Integer; cdecl;
var
  Lib: HMODULE;
  Test: TTestProc;
begin
  Lib := LoadLibrary('MSSample.dll');
  Win32Check(Lib <> 0);
  try
    try
      Test := GetProcAddress(Lib, '?fnMSSample@@YAHXZ');
      Win32Check(Assigned(Test));
      Test;
    except
      Application.HandleException(Sender);
    end;
  finally
    FreeLibrary(Lib);
  end;
end;
```

This sample DLL is compiled by Microsoft Visual Studio. It contains one exported function (fnMSSample) which calls some internal functions (Test and InternalTest) and raises access violation exception. It is loaded and called by the Delphi project (.exe host).

You should enable EurekaLog for host application. You should do this in the same way as you do it for typical application without any DLLs. For example, if you have VCL Forms application as the host - then you need to enable EurekaLog for host application and set application type to "VCL Forms Application". This will add EurekaLog code and data into final .exe file. It would also set hook for `Forms.TApplication.HandleException` method, which will allow to automatically handle exceptions without writing any code.

Important note: Please note that above example is not recommended to use in real projects. The example above is created to illustrate differences in debug information for DLLs. It is not intended to illustrate DLL design principles. The problem with example design is as following: the sample DLL does not contain any try/catch handlers, and it lets exceptions escape DLL to the caller. This is usually a bad practice 457 - because the caller may not know how to work with exceptions coming from other programming language. The above example uses hardware exception for illustration. Real-life application will probably raise software exceptions - which are specific to programming language. Thus, a better idea is to wrap fnMSSample function into try/catch block and convert exception into safe error code (a simple flag, integer code, HRESULT, etc.) - as explained in this article 467.

Since we're going to pass exceptions from DLL to .exe (a not recommended way, but sufficient for our example) - you have to do the following:

- Disable "Capture stack only for exceptions from current module" option 237. This will instruct EurekaLog to catch exceptions from any executable module. By default EurekaLog captures only exceptions within the same module.
- Disable chained exceptions support 573 by setting all options to "Classic" position. This feature requires ability to track life time of exceptions objects. This is not possible for general case (e.g. host and DLL are compiled by different compilers and there is no assist from RTL for tracking exception objects). This feature *may* work in some specific configurations.

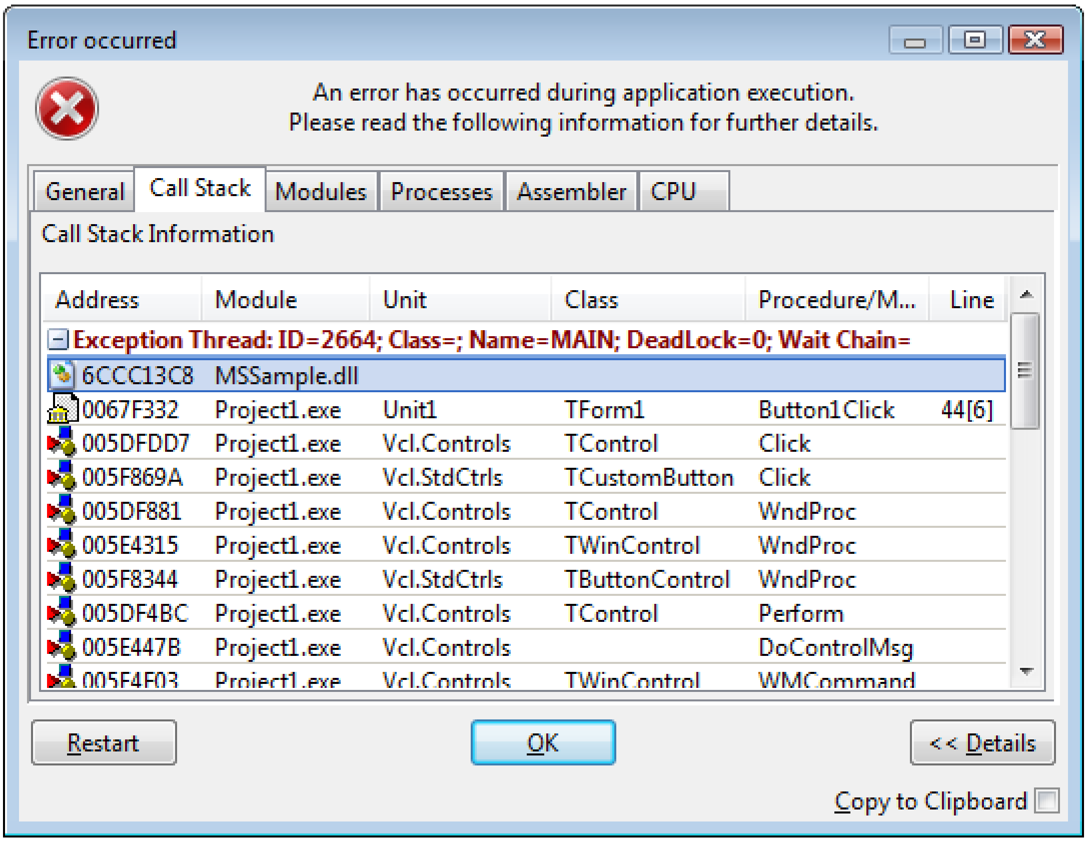
You don't need to perform these changes if you're using the recommended approach 457 of not letting exceptions escape DLL. See example here 467.

Note: notice that event handler for Button1 in the above example calls exception handler (Application.HandleException) explicitly. This is a required action for such code. That is because exceptions from DLL will be handled in default application handler without such explicit call - which happens after DLL will be unloaded. Therefore, an execution will go such way without explicit try/except block:

- Load DLL
- Call function from DLL
- Raise exception
- Unload DLL
- Analyze exception from (already unloaded) DLL

The last step will fail because DLL was already unloaded. This is the reason behind explicit call to Application.HandleException. Please note that you don't need such call if you use different buttons to load DLL/call function/unload DLL - that is because default exception handler will guard each call of event handler.

This sample will generate the following error dialog without any additional actions:



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/M...	Line
Exception Thread: ID=2664; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll				
0067F332	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005DFDD7	Project1.exe	Vcl.Controls	TControl	Click	
005F869A	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005DF881	Project1.exe	Vcl.Controls	TControl	WndProc	
005E4315	Project1.exe	Vcl.Controls	TWinControl	WndProc	
005F8344	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005DF4BC	Project1.exe	Vcl.Controls	TControl	Perform	
005E447B	Project1.exe	Vcl.Controls		DoControlMsg	
005F4F03	Project1.exe	Vcl.Controls	TWinControl	WMCommand	

Exception dialog for DLL without any debug information support

As you can see: the call stack lists every function within host .exe file - because host .exe has EurekaLog debug information. DLL have no debug information. Therefore, EurekaLog is unable to show call stack for DLL. Empty first line is exact exception address. It is shown always - regardless of debug information available.

Of course, this is not a very useful bug report for DLL. You want to see some functions from that DLL.

There are three possible usage cases for non-Embarcadero DLLs:
1. Convert 3rd party debug information format into EurekaLog debug information;
2. Use 3rd party debug information without converting to EurekaLog debug information;
3. Use DLL exports provider (no debug information is available or debug information format is not supported).

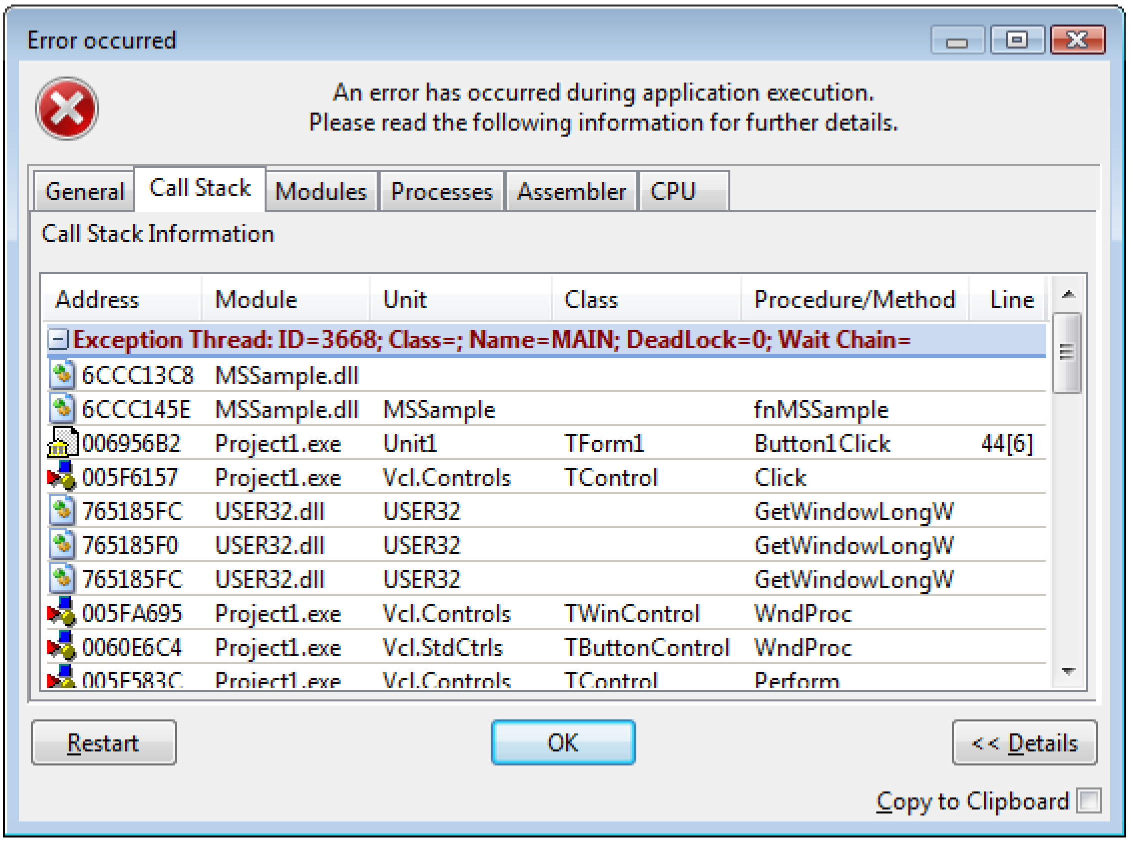
Use DLL exports provider (no debug information is available or debug information format is not supported)

This approach is not recommended - because there will be inaccurate information in call stacks without debug information. This case should be used only if compiler is not able to produce debug information or debug information format is unknown to EurekaLog. Use this approach as "last resort" measure: to show at least something for DLLs without debug information.

1. Enable DLL exports debug information provider 355 (see description of this provider 411).

Note: this case is a default configuration for EurekaLog.

The result error message will look like this:



Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
⊟ Exception Thread: ID=3668; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll				
6CCC145E	MSSample.dll	MSSample		fnMSSample	
006956B2	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F6157	Project1.exe	Vcl.Controls	TControl	Click	
765185FC	USER32.dll	USER32		GetWindowLongW	
765185F0	USER32.dll	USER32		GetWindowLongW	
765185FC	USER32.dll	USER32		GetWindowLongW	
005FA695	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060E6C4	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F583C	Project1.exe	Vcl.Controls	TControl	Perform	

Exception dialog for DLL without debug information but with DLL Export provider

This example is able to discover name of exported function ('fnMSSample') - thanks to DLL Export provider. However, this example is not able to identify internal functions in DLL - because internal functions are not exported. Therefore, internal functions are not listed at all. And (as always) exact exception location is added to the top of the call stack.

Please note that this example also adds entries for USER32.dll and KERNEL32.dll in the call stack.

Note: DLL Exports provider may show entries like "(possible fnMSSample+132)". Such text means that there are some JMP or RET instructions between start of the function and actual address in a call stack. This means:

- [Positive] Address belongs to the specified function. JMP/RET instruction may be part of the function's logic (such as try/except block);
- [False-positive] Address does not belong to the specified function. JMP/RET instruction marks the end of the function. Address itself lies within some other internal/unknown function after the specified function.

Number after "+" sign indicate byte offset between function's start and call stack address. Greater offsets usually indicate greater chance for false-positive entries.

Use 3rd party debug information without converting to EurekaLog debug information

This approach can be used if you want to use other tools for your executable (for example: Process Explorer, WinDBG, external debugger, etc.). Other tools are not able to recognize and read EurekaLog debug information. Thus, you need to supply and keep debug information in a known format - such as PDB/DBG, TD32, etc. Both EurekaLog and other tools will be able to use this debug information.

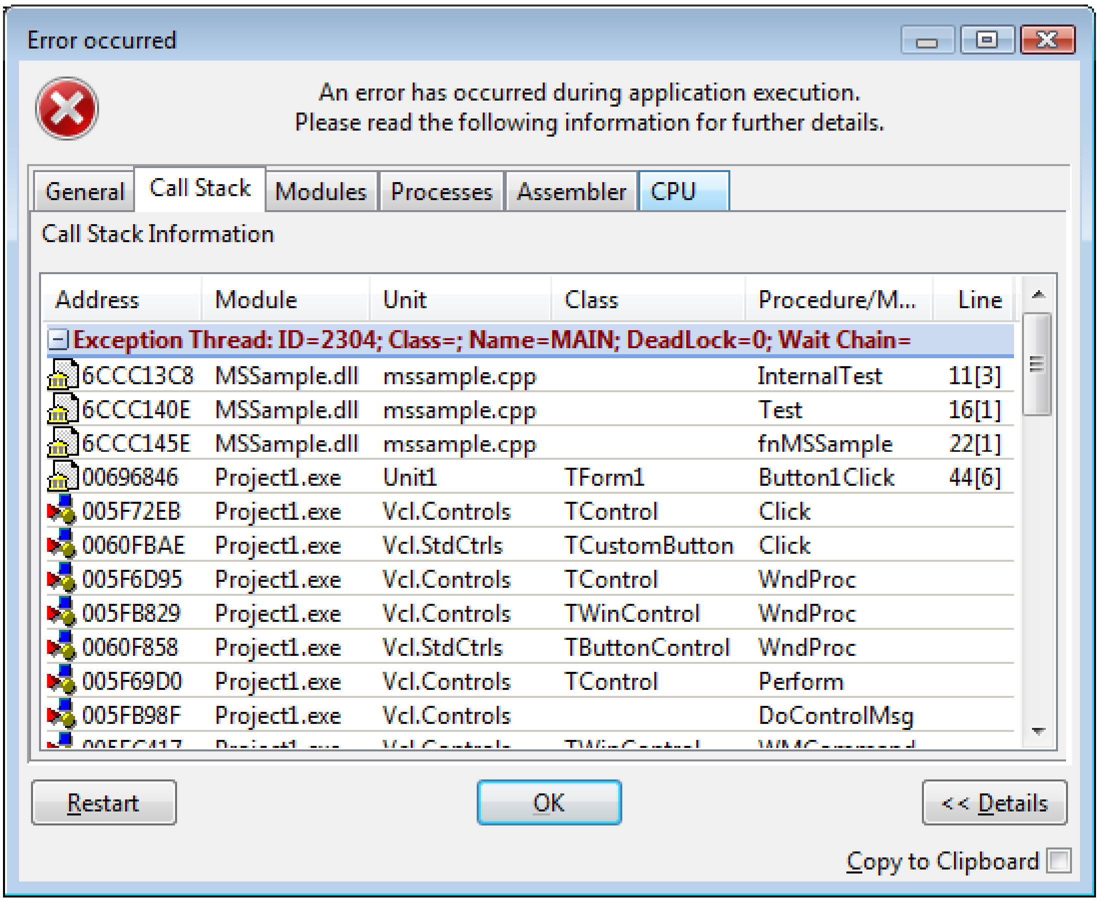
1. Enable generation of debug information in project's options (see below);
2. Enable support for debug information format 355 in EurekaLog's project options (see list of supported formats 409).

For example, we use Microsoft Visual Studio in the above example. You can go to "Project" / "Properties" IDE menu item to open options for your C++ DLL project. Go to "Configuration Properties" / "Linker" / "Debugging" and enable "Generate debug info" option. Go to Go to "Configuration Properties" / "C/C++" / "General" and set "Debug Information Format" option option to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler). Build your project. There should be .pdb file available in the same folder as .dll file for your project.

You should deploy this .pdb file with your .dll file.

Enable support for PDB debug information format by enabling "Microsoft Dbg/PDB" option 355 .

Note: you can enable generation of .map files in your Visual Studio projects. However, such files can not be used by EurekaLog. .map files do not have a strict format. .map files are defined as "human-readable plain text files in free form that indicate the relative offsets of functions for a given version of a compiled binary". EurekaLog is able to parse .map files produced by Delphi and C++ Builder linkers. EurekaLog is not able to parse .map files produced by other compilers/linkers/tools.


Error occurred

An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/M...	Line
Exception Thread: ID=2304; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll	mssample.cpp		InternalTest	11[3]
6CCC140E	MSSample.dll	mssample.cpp		Test	16[1]
6CCC145E	MSSample.dll	mssample.cpp		fnMSSample	22[1]
00696846	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F72EB	Project1.exe	Vcl.Controls	TControl	Click	
0060FBAE	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005F6D95	Project1.exe	Vcl.Controls	TControl	WndProc	
005FB829	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060F858	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F69D0	Project1.exe	Vcl.Controls	TControl	Perform	
005FB98F	Project1.exe	Vcl.Controls		DoControlMsg	
005FC417	Project1.exe	Vcl.Controls	TWinControl	WMCommand	

Exception dialog for DLL with .pdb file and enabled MS Debug Info provider

Since DLL now have full debug information available in .pdb file, and EurekaLog has enabled support for reading .pdb files - there will be full information for your DLL in the call stack. All exported and internal functions will be properly identified. All functions will have line numbers information.

Note:

- **IMPORTANT: (only for "Microsoft Dbg/PDB" provider)** You have to deploy .pdb file with your DLL. Unfortunately, PDB information can not be injected into executable. Only standalone .pdb files are supported. This is not a limitation of EurekaLog;
- **IMPORTANT: (only for "Microsoft Dbg/PDB" provider)** You have to deploy DbgHelp.dll

file from Microsoft Debugging Tools. This file can also be found in \Bin (\Bin64) folder of EurekaLog installation. Default DbgHelp.dll from C:\Windows is not suitable for such usage.

- You can use any other source of debug information: such as TD32 (.tds), MAP (.map), DBG (.dbg), JDBG (.jdbg), etc. Just be sure to enable corresponding debug information provider 355. Most other debug formats can be injected into executable and not require any helper DLLs;
- You can use PE Analyzer (Module Informer) tool 617 to check whenever DLL has any supported debug information;
- You can also enable debug information for Windows DLLs to show precise information about internal functions in system DLLs. This configuration is explained here 504.

Convert 3rd party debug information format into EurekaLog debug information

.pdb files are analog of .tds files (with TD32 debug information): these files are extremely large (debug information file could be more than 10x times larger than executable module itself), binary, uncompressed, unencrypted, store huge amount information about executable (functions, arguments, types, classes, scopes, line numbers, etc. - in other words, all information that may be needed for the debugger). Since exception tracer does not need all this information (units/routine names and line numbers are enough) - obviously, deploying such files along with your DLLs is not a best solution. Surely, you have to use .pdb files if you need to load your DLL into other tools (such as Process Explorer or WinDBG), but if you just want to use exception tracer tool with your DLL - there must be a better way.

A better way is to convert 3rd party debug information into EurekaLog debug information. EurekaLog debug information is compact, compressed, encrypted and stores only minimum amount of information necessary to build call stack. All other extra information is not stored. And you won't need any external helper DLLs - like DbgHelp.dll.

This is recommended approach. You can convert some supported debug information formats into EurekaLog debug information format. You can do this without DLL recompilation 426.

1. Enable generation of debug information in project's options (the same as in the previous approach - see above);
2. Compile your DLL. There will be .dll file and debug information file (such as .pdb);
3. Run ecc32.exe or emake.exe to post-process your DLL file with embedding EurekaLog debug information 38 (see below).

Ecc32/emake tools can use --el_alter_exe command line switch to specify target .dll file for post-processing (you should use NUL as project file name for --el_alter_exe switch - since your DLL is not a Delphi / C++ Builder project), --el_config switch to specify EurekaLog configuration (you have to use external .eof file since there is no Delphi / C++ Builder project to read configuration from), --el_source to specify debug information source (default source is Delphi / C++ Builder .map files; you have to specify where ecc32/emake should look for debug information).

Notes:

- You have to create new .eof file 443 which will contain EurekaLog configuration for your DLL. This file is required since there is no Delphi / C++ Builder project (which usually stores EurekaLog configuration).
- Most options in .eof file will be ignored since there is no EurekaLog code in your DLL. Only design-time / build options will be used: such as encryption for debug information, stripping relocs, removing function names, etc.
- You can enable generation of .map files in your Visual Studio projects. However, such files can not be used by EurekaLog. .map files do not have a strict format. .map files are defined as "human-readable plain text files in free form that indicate the relative offsets of functions for a given version of a compiled binary". EurekaLog is able to parse .map files produced by Delphi and C++ Builder linkers. EurekaLog is not able to parse .map files produced by other compilers/linkers/tools. Therefore, only Delphi / C++ Builder .map files could be used for post-processing. Other possible source includes .tds files (TD32), any format that DbgHelp supports (usually: .pdb and .dbg files).

- PDB format is a modern debug information format for Microsoft Visual Studio tool chain. It can contain much more information than older DBG debug information format. DBG format support is limited in many tools. For example. It is recommended to use PDB format when possible.

For example, we have Microsoft Visual Studio DLL project as described above. Project have some internal function and exports one function (fnMSSample). You can go to "Project" / "Properties" IDE menu item to open options for your C++ DLL project. Go to "Configuration Properties" / "Linker" / "Debugging" and enable "Generate debug info" option. Go to Go to "Configuration Properties" / "C/C++" / "General" and set "Debug Information Format" option option to "Program Database" (/Zi option for ompiler) or "Program Database for Edit And Continue" (/ZI option for compiler). Build your project. There should be .pdb file available in the same folder as .dll file for your project.

Now, run ecc32.exe or emake.exe with the following command-line:

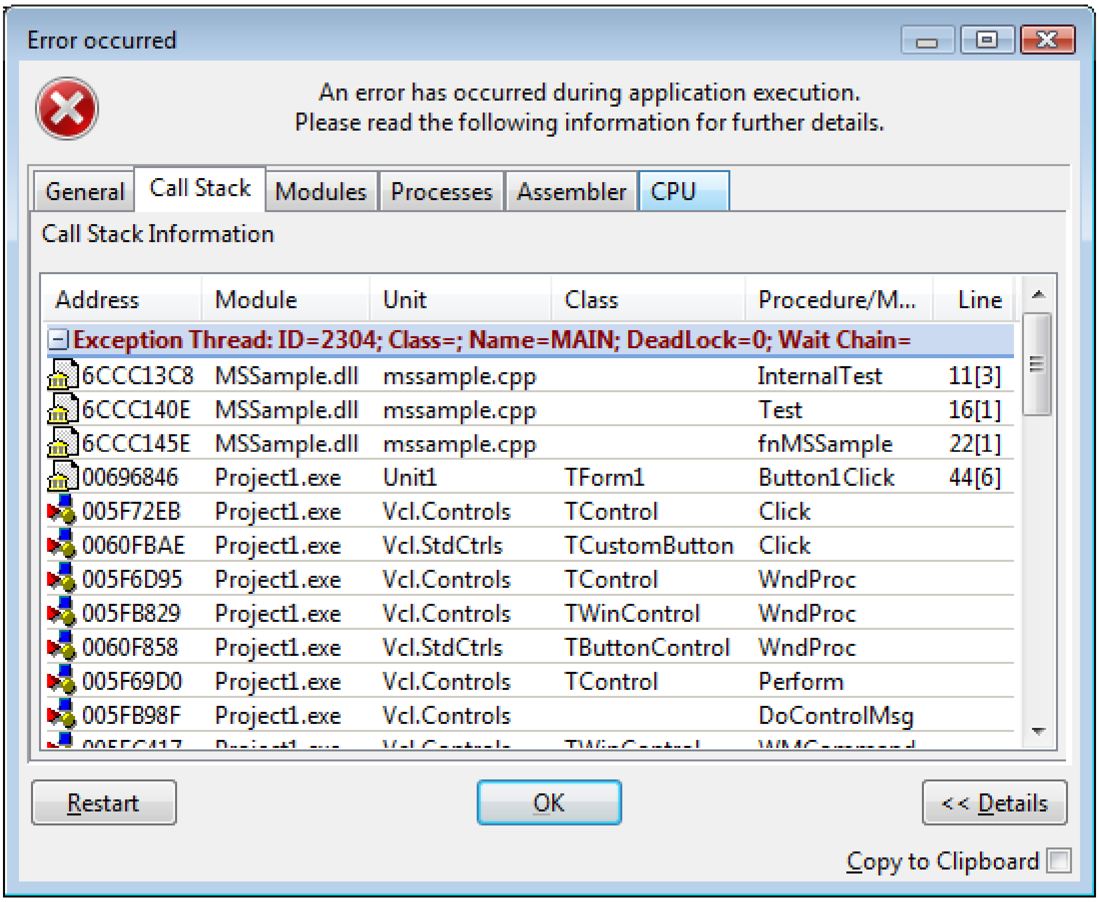
```
"ecc32.exe"  "--el_alter_exe=NUL;MSSample.dll"  "--el_config=MSSample.eof"  --el_source=PDB
```

(you may need to specify full or relative file paths for your files; do not forget to enclose file paths with spaces in double quotes)

This command line will convert MSSample.pdb file into EurekaLog format and inject this information into MSSample.dll file. Options for this operation are specified in MSSample.eof file.

Note: MSSample.pdb file may be deleted after conversion - depending on the state of "Delete service files after compilation" option 234.

Resulting DLL will have injected EurekaLog debug information - which could be verified by using PE Analyzer (Module Informer) tool 617. No additional debug information providers should be enabled. The exception dialog will look like this (the same as in the previous approach):



Address	Module	Unit	Class	Procedure/M...	Line
⊟ Exception Thread: ID=2304; Class=; Name=MAIN; DeadLock=0; Wait Chain=					
6CCC13C8	MSSample.dll	mssample.cpp		InternalTest	11[3]
6CCC140E	MSSample.dll	mssample.cpp		Test	16[1]
6CCC145E	MSSample.dll	mssample.cpp		fnMSSample	22[1]
00696846	Project1.exe	Unit1	TForm1	Button1Click	44[6]
005F72EB	Project1.exe	Vcl.Controls	TControl	Click	
0060FBAE	Project1.exe	Vcl.StdCtrls	TCustomButton	Click	
005F6D95	Project1.exe	Vcl.Controls	TControl	WndProc	
005FB829	Project1.exe	Vcl.Controls	TWinControl	WndProc	
0060F858	Project1.exe	Vcl.StdCtrls	TButtonControl	WndProc	
005F69D0	Project1.exe	Vcl.Controls	TControl	Perform	
005FB98F	Project1.exe	Vcl.Controls		DoControlMsg	
005FC417	Project1.exe	Vcl.Controls	TWinControl	WMCommand	

Exception dialog for DLL with injected EurekaLog debug information

This call stack will be the same as in the previous approach: DLL now have full debug information which is injected into DLL file - there will be full information for your DLL in the call stack. All exported and internal functions will be properly identified. All functions will have line numbers information.

The difference from the previous approach is that you don't need to deploy any additional files along with your DLL. All necessary debug information is stored inside DLL file itself.

See also:
- Using EurekaLog with DLLs post-processed by 3rd party tools (JCL, madExcept, etc.) 495
- Post-processing without (re)compilation 426
- Working with configuration 439
- External configuration and .eof files 443
- PE Analyzer (Module Informer) tool 617

11.7 System logging setup

System Log can be used by applications to view and report different events. System Log is often used by Win32 service applications. However, any kind of application can use System Log.

Note: while it is recommended to use System Log for Win32 service applications, it is not strictly required. Win32 service applications can use typical EurekaLog bug report files 46 instead of System Log.

This topic contains the following subsections:
- What is System Log 536
- Registering Event Source 537
- Configuring error dialog 543

- Using dynamic content with System Log 543

Note: you can look at NT Service Application demo which is shipped with EurekaLog.

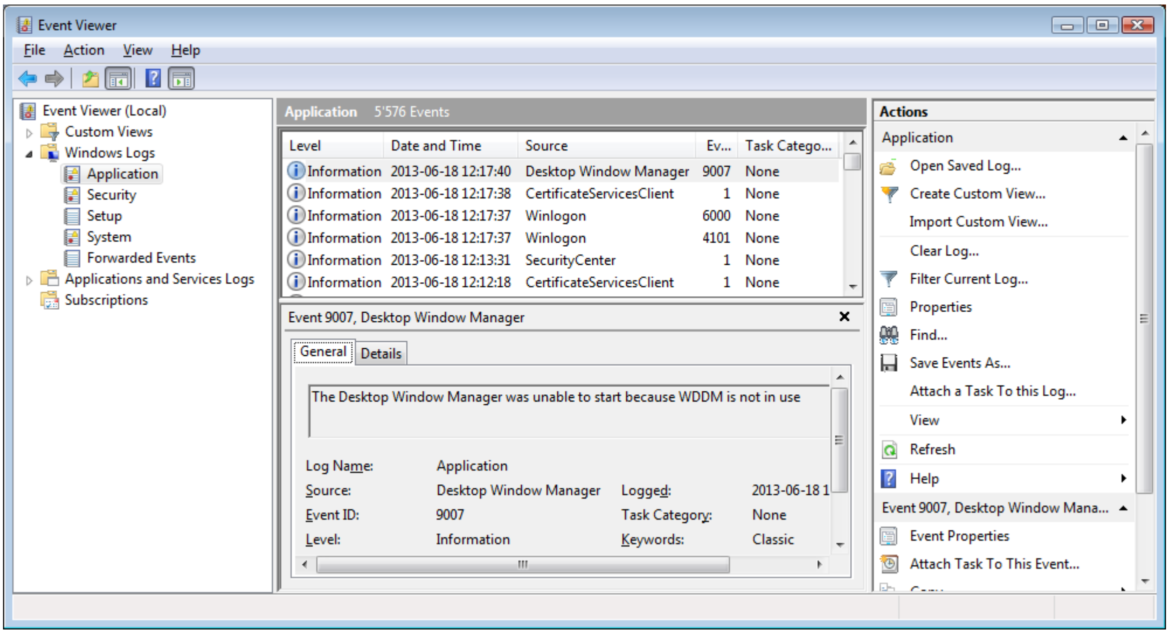
See also:

- Configuring bug report 46

11.7.1 System Log

Many applications record errors and events in proprietary error logs, each with their own format and user interface. Data from different applications can't easily be merged into one complete report, requiring system administrators or support representatives to check a variety of sources to diagnose problems.

Event logging provides a standard, centralized way for applications (and the operating system) to record important software and hardware events. The event logging service records events from various sources and stores them in a single collection called an event log. The Event Viewer enables you to view logs; the programming interface also enables you to examine logs.



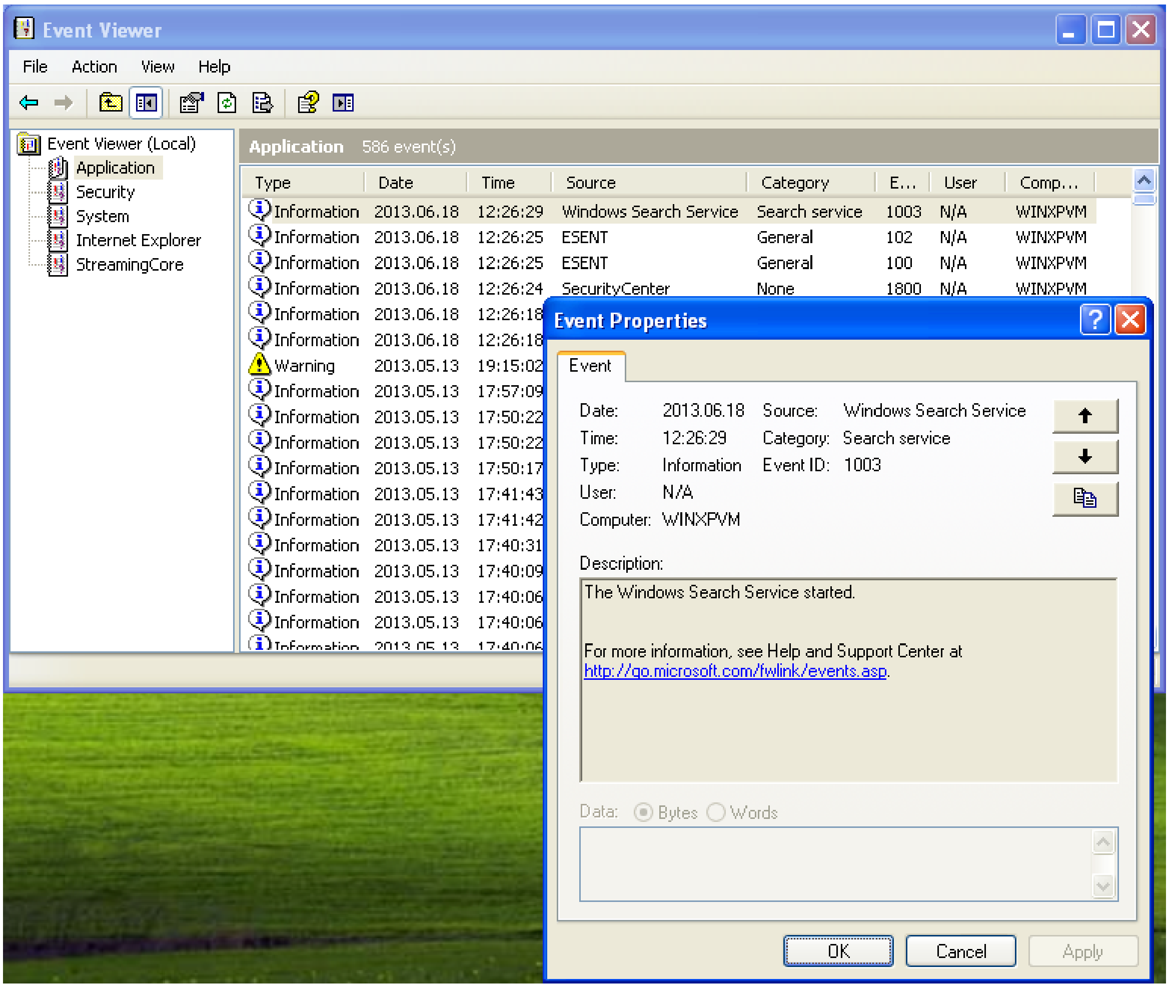
Level	Date and Time	Source	Ev...	Task Catego...
Information	2013-06-18 12:17:40	Desktop Window Manager	9007	None
Information	2013-06-18 12:17:38	CertificateServicesClient	1	None
Information	2013-06-18 12:17:37	Winlogon	6000	None
Information	2013-06-18 12:17:37	Winlogon	4101	None
Information	2013-06-18 12:13:31	SecurityCenter	1	None
Information	2013-06-18 12:12:18	CertificateServicesClient	1	None

Event 9007, Desktop Window Manager

The Desktop Window Manager was unable to start because WDDM is not in use

Log Name:	Application		
Source:	Desktop Window Manager	Logged:	2013-06-18 1
Event ID:	9007	Task Category:	None
Level:	Information	Keywords:	Classic

Event Viewer in Windows Vista+



Type	Date	Time	Source	Category	E...	User	Comp...
Information	2013.06.18	12:26:29	Windows Search Service	Search service	1003	N/A	WINXPVM
Information	2013.06.18	12:26:25	ESENT	General	102	N/A	WINXPVM
Information	2013.06.18	12:26:25	ESENT	General	100	N/A	WINXPVM
Information	2013.06.18	12:26:24	SecurityCenter	None	1800	N/A	WINXPVM
Information	2013.06.18	12:26:18					
Information	2013.06.18	12:26:18					
Warning	2013.05.13	19:15:02					
Information	2013.05.13	17:57:09					
Information	2013.05.13	17:50:22					
Information	2013.05.13	17:50:22					
Information	2013.05.13	17:50:17					
Information	2013.05.13	17:41:43					
Information	2013.05.13	17:41:42					
Information	2013.05.13	17:40:31					
Information	2013.05.13	17:40:09					
Information	2013.05.13	17:40:06					
Information	2013.05.13	17:40:06					
Information	2013.05.13	17:40:06					

Date: 2013.06.18 Source: Windows Search Service
Time: 12:26:29 Category: Search service
Type: Information Event ID: 1003
User: N/A
Computer: WINXPVM

Description:

The Windows Search Service started.

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

Event Viewer in Windows XP

Applications can use the Event Logging API to report and view events. While any application can log events with Event Logging API, Win32 service applications use Event Logging most commonly.

Note: while it is recommended to use System Log for Win32 service applications, it is not strictly required. Win32 service applications can use typical EurekaLog bug report files 46 instead of System Log.

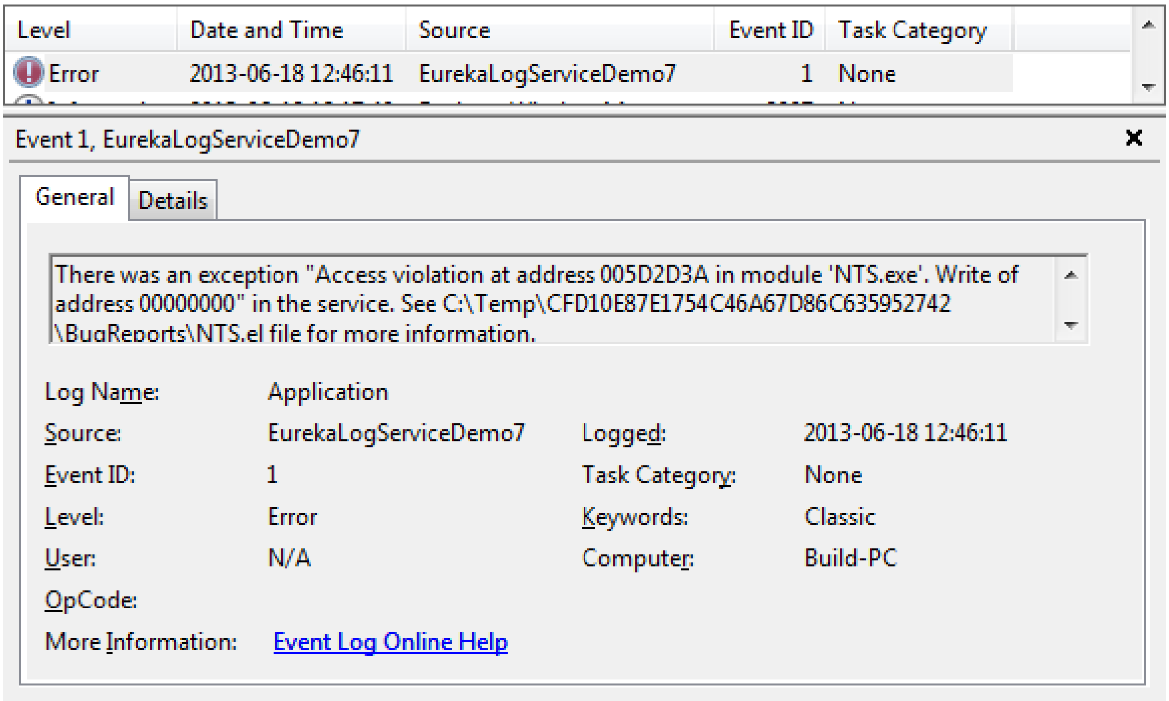
See also:

- Registering Event Source 537
- Configuring bug report 46
- System Logging setup 535
- System Log dialog 384

11.7.2 Registering Event Source

Before your application can add events to system log - you must to register your application as source. If you fail to do this, then system will not be able to display your events correctly.

The event source is the name of the software that logs the event. It is often the name of the application or the name of a subcomponent of the application if the application is large. You can add a maximum of 16,384 event sources to the registry.



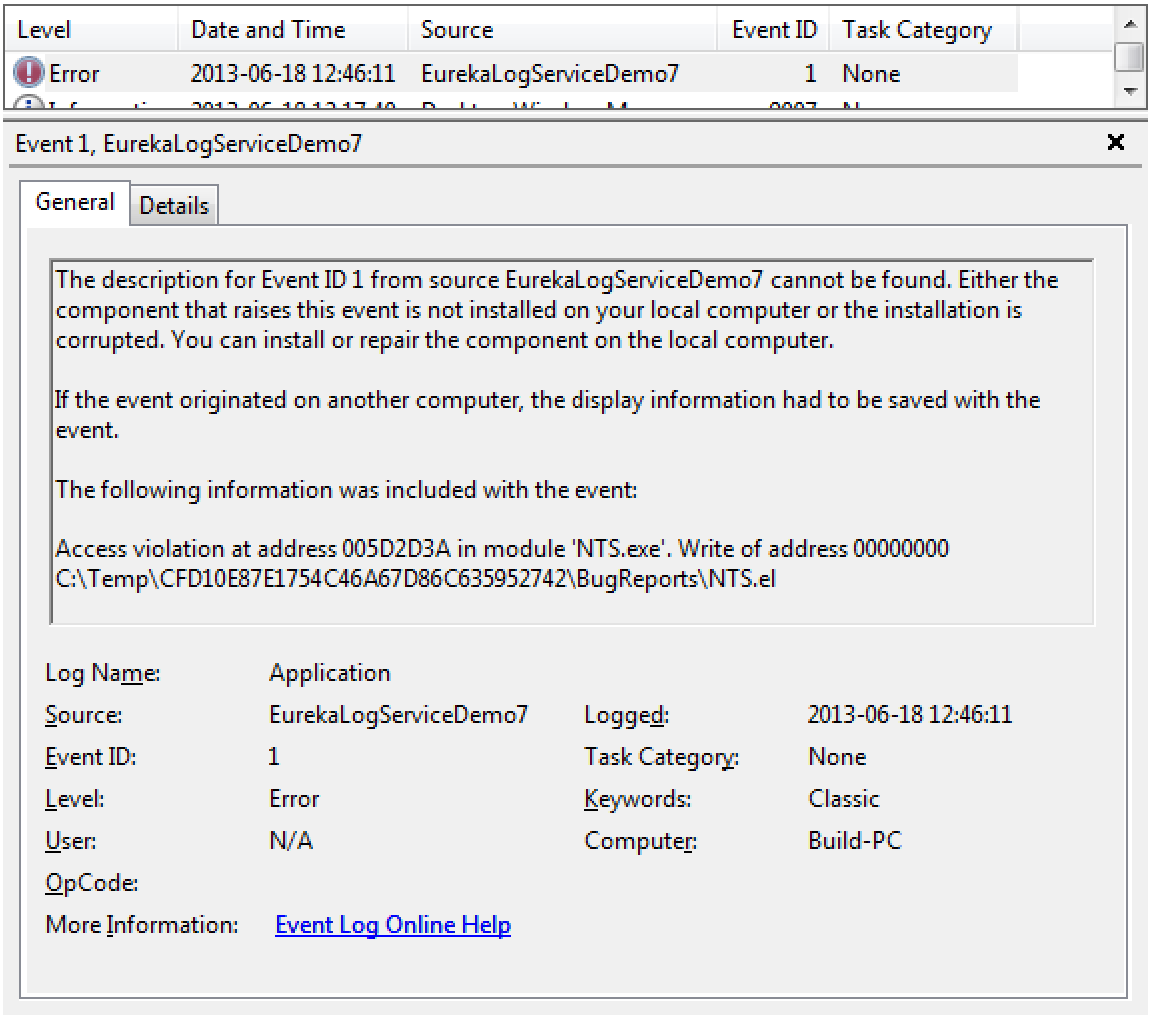
Level	Date and Time	Source	Event ID	Task Category
Error	2013-06-18 12:46:11	EurekaLogServiceDemo7	1	None

Event 1, EurekaLogServiceDemo7

There was an exception "Access violation at address 005D2D3A in module 'NTS.exe'. Write of address 00000000" in the service. See C:\Temp\CFD10E87E1754C46A67D86C635952742\BugReports\NTS.el file for more information.

Log Name: Application
Source: EurekaLogServiceDemo7 Logged: 2013-06-18 12:46:11
Event ID: 1 Task Category: None
Level: Error Keywords: Classic
User: N/A Computer: Build-PC
OpCode:
More Information: Event Log Online Help

Event Viewer displays an event from the properly registered event source



Level	Date and Time	Source	Event ID	Task Category
Error	2013-06-18 12:46:11	EurekaLogServiceDemo7	1	None

Event 1, EurekaLogServiceDemo7

The description for Event ID 1 from source EurekaLogServiceDemo7 cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer.

If the event originated on another computer, the display information had to be saved with the event.

The following information was included with the event:

Access violation at address 005D2D3A in module 'NTS.exe'. Write of address 00000000
C:\Temp\CFD10E87E1754C46A67D86C635952742\BugReports\NTS.el

Log Name: Application
Source: EurekaLogServiceDemo7 Logged: 2013-06-18 12:46:11
Event ID: 1 Task Category: None
Level: Error Keywords: Classic
User: N/A Computer: Build-PC
OpCode:
More Information: Event Log Online Help

The same event for unregistered event source

Preparing .res file for your application

To report events, you must first define your events in a message text file. Usually, you need only one event (its type will be "error") for all exceptions in your application. However, nothing stops you from having multiple events: you may want to use some "informational" or "warning" events for non-fatal/known exceptions. See this article for tips on construction of your message texts: Logging Guidelines.

The following example shows a sample message text file (Messages.mc):

```
MessageIdTypedef=DWORD
SeverityNames=(Success=0x0:STATUS_SEVERITY_SUCCESS
               Informational=0x1:STATUS_SEVERITY_INFORMATIONAL
               Warning=0x2:STATUS_SEVERITY_WARNING
               Error=0x3:STATUS_SEVERITY_ERROR
              )
FacilityNames=(System=0x0:FACILITY_SYSTEM
               Runtime=0x2:FACILITY_RUNTIME
               Stubs=0x3:FACILITY_STUBS
               Io=0x4:FACILITY_IO_ERROR_CODE
              )
LanguageNames=(English=0x409:MSG00409)
LanguageNames=(Italian=0x410:MSG00410)
MessageId=0x1
Severity=Error
Facility=Runtime
SymbolicName=mcEurekaLogErrorMessage
Language=English
There was an exception "%1" in the service.
See %2 file for more information.
.
Language=Italian
C'e stato un eccezione "%1" nel servizio.
Vedere il file %2 per ulteriori informazioni.
.
```

Warning: there must be line break at the last line with dot (.) in .mc file. In other words: you should add a blank line at the end of the file.

This file defines one event with ID #1, which is represented in two languages: English and Italian. This event has error type and uses two variable parts 543 (%1 and %2) in its error text.

To compile the Unicode message text file, use the following command:

```
mc -U Messages.mc
```

MC.exe is Message Compiler tool, which is included with Visual Studio (Express edition does not include MC tool) or Windows SDK for Windows 7 (Windows SDK for Windows 8 does not include command line tools). Compiling .mc file will give you a C++ header file (.h file). Example of content from .h file for the above example of .mc file:

```
//
//  Values are 32 bit values laid out as follows:
//
//   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//  +---+-+-+-----------------------+-------------------------------+
//  |Sev|C|R|     Facility          |               Code            |
//  +---+-+-+-----------------------+-------------------------------+
//
//  where
//
//      Sev - is the severity code
//
//          00 - Success
//          01 - Informational
//          10 - Warning
//          11 - Error
//
//      C - is the Customer code flag
//
//      R - is a reserved bit
//
//      Facility - is the facility code
//
//      Code - is the facility's status code
//
//
// Define the facility codes
//
#define FACILITY_SYSTEM                  0x0
#define FACILITY_STUBS                   0x3
#define FACILITY_RUNTIME                 0x2
#define FACILITY_IO_ERROR_CODE           0x4


//
// Define the severity codes
//
#define STATUS_SEVERITY_WARNING          0x2
#define STATUS_SEVERITY_SUCCESS          0x0
#define STATUS_SEVERITY_INFORMATIONAL    0x1
#define STATUS_SEVERITY_ERROR            0x3


//
// MessageId: mcEurekaLogErrorMessage
//
// MessageText:
//
// There was an exception "%1" in the service.
// See %2 file for more information.
//
#define mcEurekaLogErrorMessage          ((DWORD)0xC0020001L)
```

You can safely delete this file (you can also look inside to analyze if everything had gone as expected). Note the constant for mcEurekaLogErrorMessage ($C0020001). This is a full ident of your event. This constant includes event kind ($C for "Error") and facility ID ($2 for "Facility RunTime"), as well as event's ID itself (1).

Important: Remember this constant as you would need to enter it into your configuration 543. If you're using the sample above (severity = error, facility = runtime) then this number will always have form of $C002*XXXX* (where *XXXX* is hexadecimal representation of event's ID) .

You will also get a set of .bin files (MSG00409.bin, MSG00410.bin, etc.) - one for each language, and a .rc file - template for resource compiler. Example of .rc file for our case:

```
LANGUAGE 0x10,0x1
1 11 "MSG00410.bin"
LANGUAGE 0x9,0x1
1 11 "MSG00409.bin"
```

Next, you should compile auto-generated .rc file with the following command:

```
rc -r Messages.rc
```

RC.exe is Microsoft resource compiler tool, which can be found in the same location as MC.exe. You can also use resource compiler included with Delphi or C++ Builder IDE. Compiling .rc file will give you the .res file. You can delete .rc and .bin files after obtaining .res file.

Now you can include .res file into your application: "Project" / "Add to project" or `{$R Messages.res}` (Delphi only).

Note: instead of manually compiling .rc file - you can instruct Delphi IDE to automatically compile it into .res file. You can do this by placing the following line into your project source file:

```
{$R 'Messages.res' 'Messages.rc'}
```

Of course, this approach means that you must keep all .rc and .bin files around.

Tip: you can use the following command batch file to fully regenerate .res file from .mc file with deleting all temp/intermediate files:

```
@echo off
del Messages.res > NUL
mc -u -U Messages.mc
del Messages.h > NUL
rc -r Messages.rc
del Messages.rc > NUL
del MSG*.bin > NUL
```

(you may need to adjust paths to mc/rc tools)

Note: you can look at NT Service Application demo which is shipped with EurekaLog.

Registering resource file

Once you obtained .res file and included it into your application (typically: you include .res file into .exe file, but you can also include .res file into standalone DLL), now you can register your executable as event source. You can do this by adding a certain registry subkey, for example:

```
uses
  Registry;

procedure TServiceForm.ServiceAfterInstall(Sender: TService);
const
  EVENTLOG_AUDIT_FAILURE     = $0010;
  EVENTLOG_AUDIT_SUCCESS     = $0008;
  EVENTLOG_ERROR_TYPE        = $0001;
  EVENTLOG_INFORMATION_TYPE = $0004;
  EVENTLOG_WARNING_TYPE      = $0002;
var
  Reg: TRegIniFile;
begin
  Reg := TRegIniFile.Create(KEY_ALL_ACCESS);
  try
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    if Reg.OpenKey
       ('\SYSTEM\CurrentControlSet\Services\Eventlog\Application\' + Name,
    begin

      // Indicate where to look for message texts
      Reg.WriteString
        ('\SYSTEM\CurrentControlSet\Services\Eventlog\Application\' + Name
        'EventMessageFile', ParamStr(0));

      // Indicate which message types can be reported
      TRegistry(Reg).WriteInteger('TypesSupported',
        EVENTLOG_ERROR_TYPE or EVENTLOG_INFORMATION_TYPE or EVENTLOG_WARNI

    end;
  finally
    FreeAndNil(Reg);
  end;
end;

procedure TServiceForm.ServiceBeforeUninstall(Sender: TService);
var
  Reg: TRegIniFile;
begin
  Reg := TRegIniFile.Create(KEY_ALL_ACCESS);
  try
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Reg.EraseSection('\SYSTEM\CurrentControlSet\Services\Eventlog\Application\' + N
  finally
    FreeAndNil(Reg);
  end;
end;
```

This sample code assumes that your application is Win32 service application, so we can use Pre/Post-install events to register/unregister our application file as event source. It also assumes that messages (.res file) are included into main executable (.exe file). You can replace ParamStr(0) with another name (like resource DLL).

There are 3 standard logs (Application, System and Security) and arbitrary number of custom logs. Normally, your application would use "Application" standard log. The above code snippet assumes that. You may also use custom log for your application.

Please note that this code example uses **service name** as event source name. This name must be specified later in dialog options 543.

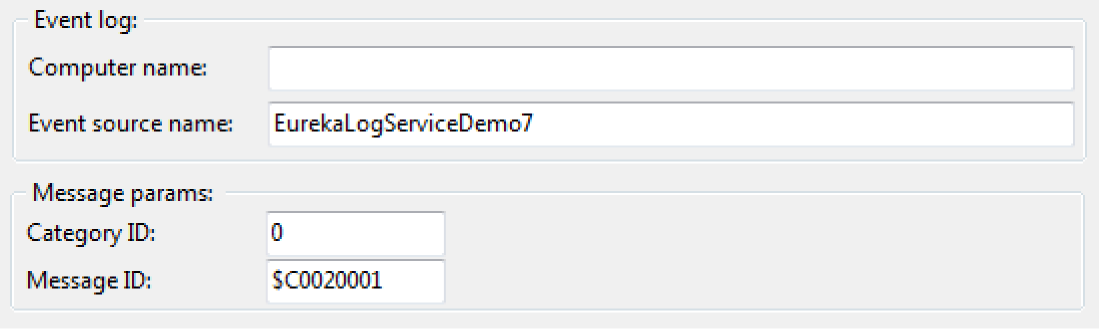
Note: you can look at NT Service Application demo which is shipped with EurekaLog.

See also:

- Configuring dialog 543
- Logging Guidelines
- System Log 536
- Using dynamic content with System Log 543

11.7.3 Configuring dialog

EurekaLog contains the special pseudo-dialog for system logging 384. You can select and configure 295 it on standard Dialogs page 267:



Event log:

Computer name:

Event source name: EurekaLogServiceDemo7

Message params:

Category ID: 0

Message ID: $C0020001

System log dialog options

Usually you should leave computer name and category ID empty (unless you want/need special behavior), enter event source name (this is the name which was used during registration 537) and message ID (this is ID which was used in message text file 537).

Important: You can get exact ID of your message from auto-generated .h file which you should obtain on the previous step 537. This ID should be full ID of your event, including severity and facility codes as well as event's ID itself.

Tip: Message ID usually have form of `$C002`*XXXX*. Where *XXXX* is ID of your event (such as 1, 2, etc.) in hexadecimal form.

Note: you can look at NT Service Application demo which is shipped with EurekaLog.

See also:

- System Log dialog options 295

11.7.4 Using dynamic content with System Log

You can write template text when you define your events. This template can include qualificators like %1, %2, etc. Qualificator defines variable part of event. Each logged event will contain reference to template's text, so template's text is not stored into log itself (thus, saving storage space). But any dynamic content for qualificators will be stored inside log. When you're viewing log - dynamic content will be inserted into qualificator's positions and form a full error message.

EurekaLog defines two qualificator by default: exception message and full path to bug report file.

You can alter default behavior to store arbitrary dynamic data. You can do this by altering send class[195]. First, you need to declare your own send class which will contain your customizations and register it within EurekaLog:

```
unit CustomizedSystemLog;

interface

implementation

uses
  Classes,
  EDialog,
  EDialogService;

type
  TEventLogDialog = class(EDialogService.TEventLogDialog)
  end;

initialization
  RegisterDialogClassFirst(TEventLogDialog);
end.
```

Next, you can alter behavior as you like. For example, default TEventLogDialog class has virtual method FillData with default implementation as follows:

```
procedure TEventLogDialog.FillData(const AData: TStrings);
begin
  AData.Add(ExceptionInfo.ExceptionMessage);
  AData.Add(LogFileName);
end;
```

which you can override to define your own dynamic content. You can use ExceptionInfo property of dialog class to get access to different properties of logged exception in question. You can also use BugReport property to gain access to bug report content.

The following example defines 4 custom dynamic pieces: %1 for exception message, %2 for exception type, %3 for compact call stack, %4 for BugID[421].

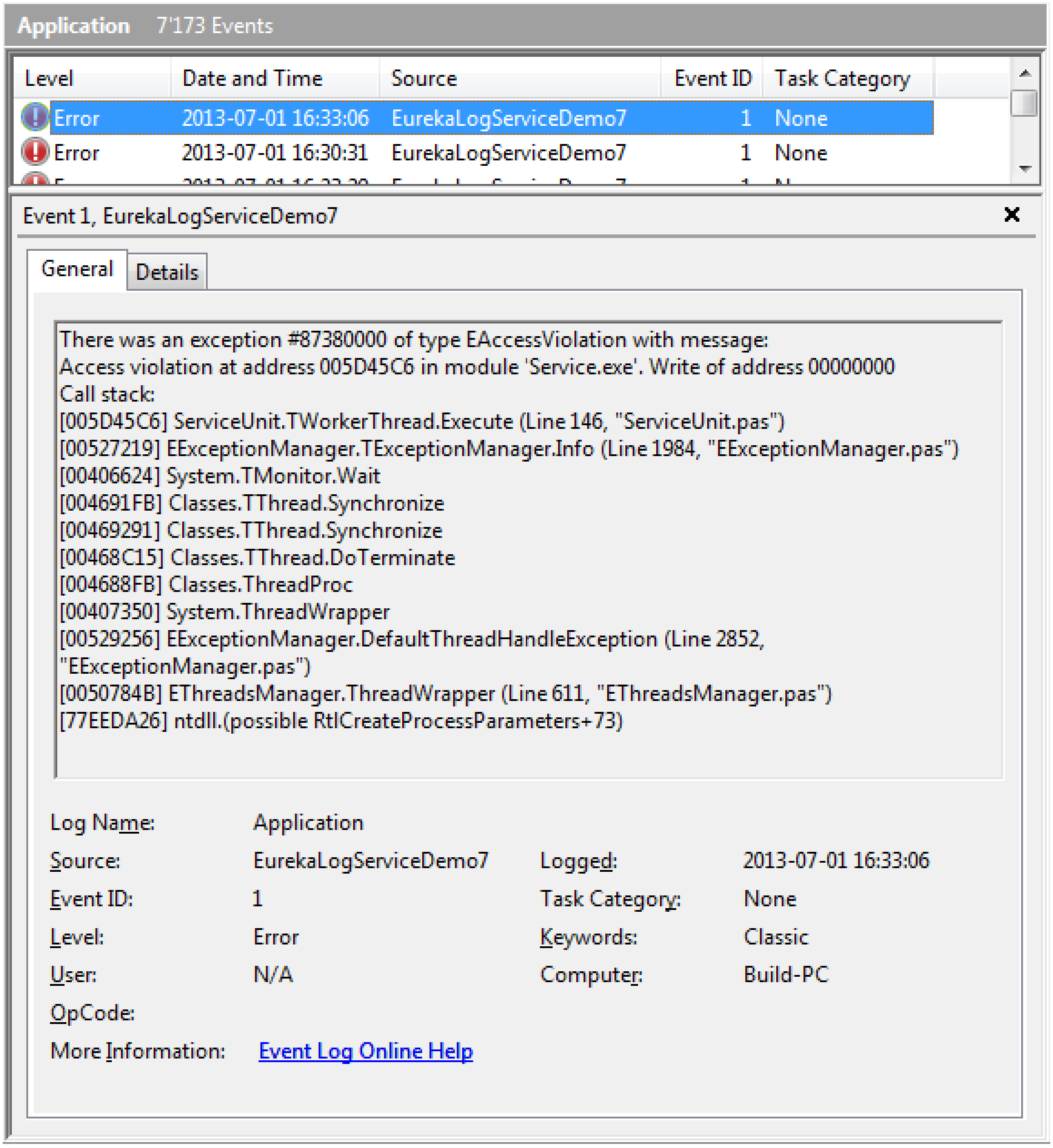
```
unit CustomizedSystemLog;

interface

implementation

uses
  SysUtils,
  Classes,
  ECallStack,
  EDialog,
  EDialogService;

type
  TEventLogDialog = class(EDialogService.TEventLogDialog)
  protected
    procedure FillData(const AData: TStrings); override;
  end;

{ TEventLogDialog }

procedure TEventLogDialog.FillData(const AData: TStrings);
var
  Formatter: TCompactStackFormatter;
  CallStack: String;
begin
  Formatter := TCompactStackFormatter.Create;
  try
    CallStack := CallStackToString(ExceptionInfo.CallStack, '', Formatter);
  finally
    FreeAndNil(Formatter);
  end;

  // %1
  AData.Add(ExceptionInfo.ExceptionMessage);

  // %2
  AData.Add(ExceptionInfo.ExceptionClass);

  // %3
  AData.Add(CallStack);

  // %4
  AData.Add(ExceptionInfo.BugIDStr);
end;

initialization
  RegisterDialogClassFirst(TEventLogDialog);
end.
```

You can use the following .mc file for this example:

```
MessageIdTypedef=DWORD
SeverityNames=(Success=0x0:STATUS_SEVERITY_SUCCESS
               Informational=0x1:STATUS_SEVERITY_INFORMATIONAL
               Warning=0x2:STATUS_SEVERITY_WARNING
               Error=0x3:STATUS_SEVERITY_ERROR
```

```
                  )
FacilityNames=(System=0x0:FACILITY_SYSTEM
               Runtime=0x2:FACILITY_RUNTIME
               Stubs=0x3:FACILITY_STUBS
               Io=0x4:FACILITY_IO_ERROR_CODE
              )
LanguageNames=(English=0x409:MSG00409)
MessageId=0x1
Severity=Error
Facility=Runtime
SymbolicName=mcEurekaLogErrorMessage
Language=English
There was an exception #%4 of type %2 with message:%n%1%nCall stack:%
n%3
.
```

The result will look like this:



Level	Date and Time	Source	Event ID	Task Category
Error	2013-07-01 16:33:06	EurekaLogServiceDemo7	1	None
Error	2013-07-01 16:30:31	EurekaLogServiceDemo7	1	None

Event 1, EurekaLogServiceDemo7

```
There was an exception #87380000 of type EAccessViolation with message:
Access violation at address 005D45C6 in module 'Service.exe'. Write of address 00000000
Call stack:
[005D45C6] ServiceUnit.TWorkerThread.Execute (Line 146, "ServiceUnit.pas")
[00527219] EExceptionManager.TExceptionManager.Info (Line 1984, "EExceptionManager.pas")
[00406624] System.TMonitor.Wait
[004691FB] Classes.TThread.Synchronize
[00469291] Classes.TThread.Synchronize
[00468C15] Classes.TThread.DoTerminate
[004688FB] Classes.ThreadProc
[00407350] System.ThreadWrapper
[00529256] EExceptionManager.DefaultThreadHandleException (Line 2852,
"EExceptionManager.pas")
[0050784B] EThreadsManager.ThreadWrapper (Line 611, "EThreadsManager.pas")
[77EEDA26] ntdll.(possible RtlCreateProcessParameters+73)
```

Log Name:	Application		
Source:	EurekaLogServiceDemo7	Logged:	2013-07-01 16:33:06
Event ID:	1	Task Category:	None
Level:	Error	Keywords:	Classic
User:	N/A	Computer:	Build-PC
OpCode:			
More Information:	Event Log Online Help		

An event with the 4 custom qualificators

Note: you can look at NT Service Application demo which is shipped with EurekaLog.

See also:

- System Log dialog options 295
- Registering Event Source 537
- Configuring error dialog 543

11.8 Multi-threaded applications

This guide for creating multithreaded applications with EurekaLog contains the following articles:

- How to create threads 547;
- EurekaLog's helpers for creating threads 547;
- Enabling EurekaLog for background threads 568;
- Multithreaded call stacks 85;
- Multithreading options 246.

See also:

- Configuring call stack 48
- Using EurekaLog with COM objects 488

11.8.1 Creating threads

There are the following possibilities to create threads in Delphi / C++ Builder applications:

- CreateThread 548 - a basic function to create any thread. This is a system function;
  - o BeginThread 549 - a wrapper for CreateThread function. This is a RTL function;
    - TThread 553 - a wrapper for BeginThread function;
      - Frameworks 565 (AsyncCalls, OmniThreadLibrary, etc.) - wrappers for TThread class;
  - o Thread pools 566 (QueueUserWorkItem, etc.) - wrappers for CreateThread function;

Usually your code will use TThread or BeginThread as low-level thread creation routines. Or your code may use frameworks/thread pools. You should never use CreateThread function to create threads.

EurekaLog offers two extended routines which greatly simplify creating threads for debugging:

- BeginThreadEx function 551 - a wrapper for BeginThread function which adds support for naming thread and enabling EurekaLog. This function can be used in any application without enabling/including EurekaLog.
- TThreadEx class 559 - a wrapper for TThread class which adds support for naming thread and enabling EurekaLog. This class can be used in any application without enabling/including EurekaLog.

It is recommended to use BeginThreadEx function instead of BeginThread function and to use TThreadEx class instead of TThread class.

Important note: turning off low-level hooks 259 means that EurekaLog will not install additional hooks for API functions. This means that EurekaLog will not intercept important system calls. For example, EurekaLog will not hook ExitThread function, which means EurekaLog will not know when a thread exits. This will lead to thread information stored forever - until application terminates. You can call internal _NotifyThreadGone or _CleanupFinishedThreads functions (from EThreadsManager unit) to notify EurekaLog about thread's termination. Such manual notifications can be avoided by using EurekaLog's wrappers (TThreadEx 559, for example).

See also:

- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.1.1 CreateThread Function

CreateThread system function is a basic function to create any thread. Any other method for creating threads is based on CreateThread function.

Warning: it is not recommended to directly use CreateThread function in your application. Always use at least BeginThread function 549 instead of CreateThread function - however, it is recommended to use BeginThreadEx function 551 instead of CreateThread. BeginThread(Ex) functions contain RTL thread support code.

Thread function for CreateThread is not protected by any exception handler. Any exception in thread created with CreateThread function will terminate the application:

- Under debugger: debugger will show "Project ... faulted with message ..." notification and terminate the application.
- Free run: system will show "... has stopped working. A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available" error dialog and terminate the application.

EurekaLog will install wrapper for any CreateThead's thread function to catch unhandled exceptions in threads (as long as "Use low-level hooks" option 259 is enabled and corresponding threading option 246 is set). Therefore you don't need to write any additional code if you're going to always use EurekaLog in your applications. EurekaLog will show a usual EurekaLog error dialog with full bug report and terminate the application.

Important note: EurekaLog has to be enabled for background threads 568.

Important note: turning off low-level hooks 259 means that EurekaLog will not install additional hooks for API functions. This means that EurekaLog will not intercept important system calls. For example, EurekaLog will not hook ExitThread function, which means EurekaLog will not know when a thread exits. This will lead to thread information stored forever - until application terminates. You can call internal _NotifyThreadGone or _CleanupFinishedThreads functions (from EThreadsManager unit) to notify EurekaLog about thread's termination. Such manual notifications can be avoided by using EurekaLog's wrappers (TThreadEx 559, for example).

However, if you want to compile your application with and without EurekaLog (for example, as release/debug builds 448) or if you don't want to install low-level hooks (for better compatibility with external tools 514) - then you have to wrap each thread function into explicit try/except block like this:

```
function ThreadFunc(Parameter: Pointer): Integer; stdcall;
begin
  try
    // 1. Name the thread for easy identification in debugger and bug reports
    NameThread('This is my thread');

    // 2. Activate EurekaLog for this thread.
    SetEurekaLogStateInThread(0, True);

    // 3. <- ... your code for the thread ...
```

```
      Result := 0; // to indicate "success"

    except

      // Handle any exception in thread
      ApplicationHandleException(nil); // from Classes unit
      Result := 1; // to indicate "failure"

    end;
  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(CreateThread(nil, 0, @ThreadFunc, nil, 0, TID));
  end;
```

Note: while most calls to VCL are not thread-safe, ApplicationHandleException function is a default global exception handler. This function is called to handle exceptions by multi-threaded applications. Therefore, this function is thread-safe. Alternative approach is to use HandleException function from EBase unit. HandleException function will show standard error dialog if EurekaLog was not enabled. HandleException function will show EurekaLog error dialog if EurekaLog was enabled. EBase unit is a special unit that can be included in any application without including full EurekaLog code.

This code sample will gracefully handles any exception in thread - regardless if EurekaLog is enabled for application or not:

- EurekaLog is enabled: this sample will show usual EurekaLog's error dialog with full bug report;
- EurekaLog is not enabled: this sample will show error message in message box.

Important note: EurekaLog has to be enabled for background threads 568.

Warning: it is not recommended to directly use CreateThread function in your application. Always use at least BeginThread function 549 instead of CreateThread function - however, it is recommended to use BeginThreadEx function 551 instead of CreateThread. BeginThread(Ex) functions contain RTL thread support code.

See also:

- BeginThread Function 549
- TThread Class 553
- Enabling EurekaLog for background threads 568

11.8.1.2 BeginThread Function

BeginThread RTL function is basic function to create threads from Delphi / C++ Builder code. This function is a simple wrapper for system CreateThread function 548.

Important note: it is recommended to use BeginThreadEx function 551 instead of BeginThread function.

BeginThread function has the following differences from CreateThread 548:

- BeginThread will make other RTL calls thread-safe (such as memory/heap operations);
- BeginThread will handle any unhandled thread exception by passing it into SysUtils.ShowException routine;

- BeginThread uses more Pascal-friendly function prototype. The functionality remains the same: there is a custom argument (pointer) and an integer return value.

There are no other differences between RTL's BeginThread and system's CreateThread functions. Therefore BeginThread is RTL's analog of system function.

Warning: you should never use CreateThread function. Always use BeginThread/ BeginThreadEx function instead of CreateThread function.

EurekaLog will install wrapper for any BeginThread's thread function to catch unhandled exceptions in threads (as long as "Use low-level hooks" option 259 is enabled). Therefore you don't need to write any additional code if you're going to always use EurekaLog in your applications. EurekaLog will show a usual EurekaLog error dialog with full bug report and terminate the application.

Important note: EurekaLog has to be enabled for background threads 568.

Important note: turning off low-level hooks 259 means that EurekaLog will not install additional hooks for API functions. This means that EurekaLog will not intercept important system calls. For example, EurekaLog will not hook ExitThread function, which means EurekaLog will not know when a thread exits. This will lead to thread information stored forever - until application terminates. You can call internal _NotifyThreadGone or _CleanupFinishedThreads functions (from EThreadsManager unit) to notify EurekaLog about thread's termination. Such manual notifications can be avoided by using EurekaLog's wrappers (TThreadEx 559, for example).

However, if you want to compile your application with and without EurekaLog (for example, as release/debug builds 448) or if you don't want to install low-level hooks (for better compatibility with external tools 514) - then you will use a default exception handling provided by BeginThread function. Any exception in threads created with BeginThread function will be handled by RTL by passing exception object to SysUtils.ShowException function. However, the default processing has the following drawbacks:

- SysUtils.ShowException function is a low-level function. Error dialog from SysUtils.ShowException is a low-level error dialog (with RAW exception address), it differs from a typical error dialog in application;
- Default exception handling terminates application after exception in thread;
- Default exception handling works in unexpected way when running outside of the debugger: there will be system error dialog followed by error dialog from SysUtils.ShowException function. There may be some other error dialogs - which are invoked during terminating application from background thread.

Therefore, a default exception handling is usually not good enough for a typical application.

Summary: you should manually catch all exceptions in thread via explicit try/except block. This is also indicated in BeginThread documentation "ThreadFunc should handle all of its own exceptions". For example:

```
function ThreadFunc(Parameter: Pointer): Integer;
begin
  try
    // 1. Name the thread for easy identification in debugger and bug reports
    NameThread('This is my thread');

    // 2. Activate EurekaLog for this thread.
    SetEurekaLogStateInThread(0, True);

    // 3. <- ... your code for the thread ...
    Result := 0; // to indicate "success"

  except

    // Handle any exception in thread
    ApplicationHandleException(nil); // from Classes unit
    Result := 1; // to indicate "failure"

  end;
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(BeginThread(nil, 0, ThreadFunc, nil, 0, TID));
end;
```

Note: while most calls to VCL are not thread-safe, ApplicationHandleException function is a default global exception handler. This function is called to handle exceptions by multi-threaded applications. Therefore, this function is thread-safe. Alternative approach is to use HandleException function from EBase unit. HandleException function will show standard error dialog if EurekaLog was not enabled. HandleException function will show EurekaLog error dialog if EurekaLog was enabled. EBase unit is a special unit that can be included in any application without including full EurekaLog code.

This code sample will gracefully handles any exception in thread - regardless if EurekaLog is enabled for application or not:

- EurekaLog is enabled: this sample will show usual EurekaLog's error dialog with full bug report;
- EurekaLog is not enabled: this sample will show error message in message box.

Important note: EurekaLog has to be enabled for background threads 568.

See also:

- CreateThread Function 548
- BeginThreadEx Function 551
- TThread Class 553
- Enabling EurekaLog for background threads 568

11.8.1.3 BeginThreadEx Function

EurekaLog provides two helper routines to simplify thread management: **BeginThreadEx** function and **TThreadEx** class - which should be used instead of BeginThread function and TThread class respectively. Both routines are from EBase unit. EBase unit is a special unit that can be included in any application without including full EurekaLog code. Therefore you can safely use EBase unit in your applications even without EurekaLog enabled.

Both BeginThreadEx function and TThreadEx class offers two additional arguments:

- Thread name;
- EurekaLog's state.

Note: the EurekaLog's state will be ignored if you compile your application without EurekaLog (or with disabled EurekaLog). On the other hand, the thread name is never ignored. Thread name is used by debugger to show thread's name in Threads window.

Thread name is arbitrary text string which can contain any combination of characters. Thread name will be shown in debugger (Threads window) and it will be used as thread caption in bug reports. You can use thread name to identificate threads.

Example of using BeginThreadEx function for this sample code 549:

```
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;
```

This code sample works exactly as the previous code sample 551. You can pass thread name right into BeginThreadEx function. Thread will be automatically named - regardless of EurekaLog's state.

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:

```
  BeginThreadEx(nil, 0, ThreadFunc, nil, 0, TID, 'Thread Name', False
{ disable EurekaLog in thread } );
```

Notes:
- Thread function for BeginTheadEx function has the same signature (prototype) as thread function for BeginThread function. I.e. you don't have to change thread function declaration.
- BeginThreadEx function has almost the same signature (prototype) as BeginThread function: the only difference is two additional arguments, which are optional. This means that you can just add "Ex" suffix to your existing BeginThread calls.
- There is no need to wrap thread function for BeginThreadEx into explicit try/except block - as you do this for thread function from BeginThread function 549. BeginThreadEx always protect thread function with exception handling block with a call to default exception handler - regardless of EurekaLog's state in this thread and your application. Any exception in thread created with BeginThreadEx function will be handled by default handler - showing either error message (if EurekaLog is not enabled) or full bug report dialog (if EurekaLog is enabled), and thread exit code will be non-zero, application will not be terminated. Use explicit try/except block if you want some custom processing for exceptions (for example, if you want to terminate application after handling exception from background thread). This also means that both thread functions (with and without try/except blocks) are fully compatible with BeginThreadEx function.
- The above facts mean that BeginThreadEx function is fully source-compatible with BeginThread function. Therefore, you can do a search&replace "BeginThread" -> "BeginThreadEx" over all source files for your project. This is a quick way to manually enable EurekaLog 570 for your threads. The same is true for TThread -> TThreadEx 559.
- BeginThreadEx function does not terminate application when thread exception is raised.

- Thread will return non zero exit code when there was exception in this thread. Otherwise (i.e. if there was no exception in the thread) - the exit code will match result of the thread function (usually - zero). This can be used to get "success/failure" exit status of the thread - via GetExitCodeThread function.

See also:

- CreateThread Function 548
- BeginThread Function 549
- TThread Class 553
- Enabling EurekaLog for background threads 568

11.8.1.4 TThread Class

TThread class is convenient wrapper for BeginThread function 549. TThread is an abstract class that enables creation of separate threads of execution in an application. Create a descendant of TThread to represent an execution thread in a multithreaded application. Each new instance of a TThread descendant is a new thread of execution. Multiple instances of a TThread derived class make an application multithreaded. Define the thread object's Execute method by inserting the code that should execute when the thread is executed.

Important note: it is recommended to use TThreadEx class 559 instead of TThread class.

TThread class handles all exceptions in thread function (a.k.a. Execute method) by saving exception into FatalException property. FatalException property can be analyzed after thread's termination.

Important note: TThread does not invoke any exception handling routine, it just merely saves exception into FatalException property. This means that TThread silently hides all exceptions by default. You have to analyze FatalException property manually.

> EurekaLog does not contain any hook for TThread - because TThread properly handles exceptions (even though it does not invoke any exception handler, but expects this from its caller). However, there is a "Auto-handle TThread exceptions" option 246. This option will call EurekaLog as default exception handler for TThread. Even though this option is a fast and convenient way to handle TThread's exception - it is not recommended to use for the following reasons:
>
> - This option will break any 3rd party code which uses TThread class and expects FatalException property to behave as usual;
> - This option have no effect if EurekaLog is not enabled in your application. Therefore you still have to write proper exception handling code if you're going to compile your application without EurekaLog;
> - This option will not work if "Use low-level hooks" option 259 is not enabled. You may want to disable this option for better compatibility with exe protectors/packers 520.
>
> Therefore it is recommended to keep "Auto-handle TThread exceptions" option turned off and use proper exception handling as dictated by TThread's design (see below).

There is at least five different ways to handle exceptions in TThread:

1. Re-raise exception in caller thread (recommended) 554;
2. Handle exception in caller thread 555;
3. Handle exception in OnTerminate event handler 556;
4. Handle exception by overriding DoTerminate method 557;
5. Handle exception with explicit try/except block 558.

Note: very old Delphi versions do not have some important features of TThread. For example, you can use only method #5 in Delphi 4 - because TThread in Delphi 4 lacks FatalException property and there is no proper exception support at all.

Important note: EurekaLog has to be enabled for background threads 568.

Important note: turning off low-level hooks 259 means that EurekaLog will not install additional hooks for API functions. This means that EurekaLog will not intercept important system calls. For example, EurekaLog will not hook ExitThread function, which means EurekaLog will not know when a thread exits. This will lead to thread information stored forever - until application terminates. You can call internal _NotifyThreadGone or _CleanupFinishedThreads functions (from EThreadsManager unit) to notify EurekaLog about thread's termination. Such manual notifications can be avoided by using EurekaLog's wrappers (TThreadEx 559, for example).

#1: Re-raise exception in caller thread (recommended)

A common practice with easy approach is a simple re-raise of FatalException in the caller thread:

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

procedure TMyThread.Execute;
begin
  // 1. Name the thread for easy identification in debugger and bug reports
  NameThread('This is my thread');

  // 2. Activate EurekaLog for this thread.
  SetEurekaLogStateInThread(0, True);

  // 3. <- ... your thread code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
  E: TObject;
begin
  // Create thread
  Thread := TMyThread.Create(False);
  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;
```

This code "analyzes" thread exception by taking it out from the thread and re-raising it into caller thread. This is an easy way to deal with exceptions from TThread. Exception will be handled by caller thread in a usual way (e.g. via already established exception handler). Note that normal execution path of the caller thread will be stopped - thus your code will not continue if there was an error in the background thread. A call stack for the exception will be call stack for the background thread - even though background thread is already terminated, and exception is re-raised.

#2: Handle exception in caller thread

Alternative approach could be the following:

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

procedure TMyThread.Execute;
begin
  // 1. Name the thread for easy identification in debugger and bug reports
  NameThread('This is my thread');

  // 2. Activate EurekaLog for this thread.
  SetEurekaLogStateInThread(0, True);

  // 3. <- ... your thread code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
  E: TObject;
begin
  // Create thread
  Thread := TMyThread.Create(False);
  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.
    // You should do this only after the thread has finished.
    if Assigned(Thread.FatalException) then
    begin
      HandleException(Thread.FatalException);
      Abort; // thread failed - stop right here
    end;

  finally
    FreeAndNil(Thread);
  end;
end;
```

We can analyze failure in-place instead of re-raising exception. We use HandleException function from EBase unit to handle exceptions.

Please note that we can not use ApplicationHandleException routine as we are going to analyze saved/non-raised exception. ApplicationHandleException routine always analyze active exception, it does not allow you to specify which exception should be analyzed. HandleException function will show standard error dialog if EurekaLog was not enabled. HandleException function will show EurekaLog error dialog if EurekaLog was enabled. EBase unit is a special unit that can be included in any application without including full EurekaLog code.

Note that we need to manually abort normal execution path (via Abort routine) to avoid continue working as if there was no error.

#3: Handle exception in OnTerminate event handler

Sometimes it is not a very convenient to use WaitFor in main thread to wait for thread's completion. You can use events for such cases:

```
type
  TMyThread = class(TThreadEx)
  private
    procedure Terminate(Sender: TObject);
  protected
    procedure Execute; override;
  end;

procedure TMyThread.Execute;
begin
  // 1. Name the thread for easy identification in debugger and bug reports
  NameThread('This is my thread');

  // 2. Activate EurekaLog for this thread.
  SetEurekaLogStateInThread(0, True);

  // 3. <- ... your thread code ...
end;
```

```
// Called within Synchronize from main thread
procedure TMyThread.Terminate(Sender: TObject);
begin
  if Assigned(FatalException) then
    HandleException(FatalException);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
begin
  Thread := TMyThread.Create(True, 'My thread');
  Thread.OnTerminate := Thread.Terminate;
  Thread.FreeOnTerminate := True;
  Thread.Start;  // = Resume for old Delphi versions
  Thread := nil; // never access thread var with FreeOnTerminate after Start
end;
```

This code will analyze thread exception by invoking exception handler for `FatalException` property. We install `OnTerminate` event handler to automatically analyze thread exception on thread's termination.

#4: Handle exception by overriding DoTerminate method

You can also override `DoTerminate` method instead of assigning `OnTerminate` handler. For example:

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

procedure TMyThread.Execute;
begin
  // 1. Name the thread for easy identification in debugger and bug reports
  NameThread('This is my thread');

  // 2. Activate EurekaLog for this thread.
  SetEurekaLogStateInThread(0, True);

  // 3. <- ... your thread code ...
end;

procedure TMyThread.DoTerminate;
begin
```

```
  inherited;
  if Assigned(FatalException) then
    HandleException(FatalException);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
begin
  Thread := TMyThread.Create(True, 'My thread');
  Thread.FreeOnTerminate := True;
  Thread.Start;  // = Resume for old Delphi versions
  Thread := nil; // never access thread var with FreeOnTerminate after Start
end;
```

This code will work in the same way as the previous example.

#5: Handle exception with explicit try/except block

Surely, you can also use previous approach with explicit try/except block 549 for TThread class:

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

procedure TMyThread.Execute;
begin
  try
    // 1. Name the thread for easy identification in debugger and bug reports
    NameThread('This is my thread');

    // 2. Activate EurekaLog for this thread.
    SetEurekaLogStateInThread(0, True);

    // 3. <- ... your thread code ...
  except
    // Handle any exception in thread
    ApplicationHandleException(nil); // from Classes unit
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
begin
  Thread := TMyThread.Create(True, 'My thread');
  Thread.FreeOnTerminate := True;
  Thread.Start;
  Thread := nil; // never access thread var with FreeOnTerminate after Start
end;
```

Important notes:

- EurekaLog has to be enabled for background threads 568.
- it is recommended to use TThreadEx class 559 instead of TThread class.

See also:

- BeginThread Function 549
- BeginThreadEx Function 551
- TThreadEx Class 559
- Anonymous threads 562
- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.1.5 TThreadEx class

EurekaLog provides two helper routines to simplify thread management: **BeginThreadEx** function and **TThreadEx** class - which should be used instead of BeginThread function and TThread class respectively. Both routines are from EBase unit. EBase unit is a special unit that can be included in any application without including full EurekaLog code. Therefore you can safely use EBase unit in your applications even without EurekaLog enabled.

Both BeginThreadEx function and TThreadEx class offers two additional arguments:

- Thread name;
- EurekaLog's state.

Note: the EurekaLog's state will be ignored if you compile your application without EurekaLog (or with disabled EurekaLog). On the other hand, the thread name is never ignored.

Thread name is arbitrary text string which can contain any combination of characters. Thread name will be shown in debugger (Threads window) and it will be used as thread caption in bug reports. You can use thread name to identificate threads.

TThreadEx class is descendant from TThread class - thus, all 5 exception handling methods for TThread class 559 is also applicable to TThreadEx class. For example:

```
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;
```

Important note: please notice the "inherited;" call in Execute method. This call is required for TThreadEx to make its work.

This code sample works exactly as the previous code sample 553. You can pass thread name right into TThreadEx's constructor. Thread will be automatically named - regardless of EurekaLog's state.

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:

```
  Thread := TMyThread.Create('Thread Name', False { disable EurekaLog in thread } );
```

Of course, you can use the same approach while supplying Suspended argument, for example:

```
  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 }  );
```

TThreadEx class also supports anonymous threads 562:

```
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 Start
  // 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 creating `OnTerminate` handler just for handling exceptions in thread is a lot of work - TThreadEx class presents a new property to handle exceptions automatically:

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

procedure TMyThread.Execute;
begin
  inherited; // <- this is required
  // ... your code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
begin
  Thread := TMyThread.Create(True, 'My thread');
  Thread.AutoHandleException := True;
  Thread.FreeOnTerminate := True;
  Thread.Start;
  Thread := nil; // never access thread var with FreeOnTerminate after Start
end;
```

This sample code will automatically invoke HandleException routine for FatalException property of your thread. AutoHandleException property is also available for anonymous threads created with TThreadEx class.

Notes:

- TThreadEx class has the same signature (prototype) for both constructor and Execute method as TThread class. I.e. you don't have to change declarations. The only difference is two additional arguments for the constructor, which are optional. This means that you can just add "Ex" suffix to your existing TThread calls.
- You must call inherited Execute method. Most thread implementation does not call inherited Execute methods.
- The above facts mean that TThreadEx class is mostly source-compatible with TThread class. Therefore, you can do a search&replace "TThread" -> "TThreadEx" over all source files for your project. However, you must additionally search for each TThread declaration and ensure that there is "inherited;" line in its Execute method. This approach can be using for manually enabling EurekaLog 570 for your threads.
- If AutoHandleException property is enabled:
  o TThreadEx does not clear FatalException property. Thus, this property still may be used to analyze "success/failure" exit status of the thread.
  o Thread will return non zero exit code when there was exception in this thread. Otherwise (i.e. if there was no exception in the thread) - the exit code will match result of the thread function (usually - zero). This can be used to get "success/failure" exit status of the thread.
- If AutoHandleException property is not enabled (default) - thread behavior will be the same as for TThread class:
  o FatalException property will store exception in thread.
  o Thread exit code will be equal to ReturnValue property.

See also:

- BeginThread Function 549
- BeginThreadEx Function 551
- TThread Class 553
- Anonymous threads 562
- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.1.6 Anonymous threads

Anonymous thread is a wrapper for TThread class which simplifies "fire-and-forget" approach. Creating new threads with bare TThread class is complex: you have to create your own thread class. And TThread is not capable of running arbitrary code, so you can't run just functions in threads. Another advantage of anonymous thread is capturing local variables.

So, for example, such code with TThread class:

```
type
  TMyThread = class(TThread)
  private
    FLocalVar: TSomeType;
  protected
    procedure Execute; override;
  public
    property LocalVar: TSomeType read FVar write FVar;
  end;

procedure TMyMotile.Execute;
begin
  // ... some code which uses LocalVar
end;

...

var
  Thread: TMyThread;
  LocalVar: TSomeType;
begin
  Thread := TMyMotile.Create(True);
  Thread.LocalVar := LocalVar;
  Thread.FreeOnTerminate := True;
  Thread.Start;
end;
```

can be replaced with shorter version of the code which uses anonymous thread:

```
var
  LocalVar: TSomeType;
begin
  TThread.CreateAnonymousThread(
    procedure;
    begin
      // ... some code which uses LocalVar
    end).Start;
end;
```

Internally, anonymous thread is implemented as TAnonymousThread class which is descendant from TThread class. Instances of TAnonymousThread are created via CreateAnonymousThread class method. The whole code for anonymous threads is very simple:

```
type
  TAnonymousThread = class(TThread)
  private
    FProc: TProc;
  protected
    procedure Execute; override;
  public
    constructor Create(const AProc: TProc);
  end;

{ TAnonymousThread }

constructor TAnonymousThread.Create(const AProc: TProc);
begin
  inherited Create(True);
  FreeOnTerminate := True;
  FProc := AProc;
end;

procedure TAnonymousThread.Execute;
begin
  FProc();
end;

...

class function TThread.CreateAnonymousThread(const ThreadProc: TProc): TThread;
begin
  Result := TAnonymousThread.Create(ThreadProc);
end;
```

Note: anonymous threads are created in suspended state, so you must run them manually with Start method. Suspended state allows you to alter thread's properties or pass additional parameters before running the thread.

Important notes:

- it is recommended to use TThreadEx.CreateAnonymousThread class method 559 instead of TThread.CreateAnonymousThread class method.
- anonymous threads are created with FreeOnTerminate property set to True. This means that you should never access thread variable after you have called Start method.

TAnonymousThread class have no additional features except being able to accept arbitrary anonymous function to be executed in its Execute method. Since anonymous threads are based on TThread class - the error handling for anonymous thread is the same as for the TThread class 553 - that is, five different methods. Obviously, you can not override DoTerminate method (since you do not declare a class), but all other exception handling methods are fully applicable for anonymous threads.

Important notes:

- EurekaLog has to be enabled for background threads 568.
- it is recommended to use TThreadEx class 559 instead of TThread class to create anonymous threads. Anonymous threads created with TThreadEx class supports AutoHandleException property 559 - which provides one more method to handle exceptions for anonymous threads.

See also:

- BeginThread Function 549
- BeginThreadEx Function 551
- TThread class 553

- TThreadEx Class 559
- Thread pools 566
- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.1.7 Frameworks

There are many different threading frameworks for Delphi and C++ Builder. Examples of threading frameworks are OmniThreadLibrary and AsyncCalls. There are few more less known threading frameworks. Also, a threading library can be included into bigger framework. For example, Indy have threading sub-library, JCL have synchronization primitives, etc.

Threading frameworks offer additional features, simplify common tasks, provide cross-platform approach, etc.

Different frameworks provide different ways to handle exceptions. You should refer to documentation for the framework. You should complete two tasks to use EurekaLog with threading framework:
1. Enable EurekaLog for threads created by threading framework;
2. Call EurekaLog exception handler for unhandled exceptions in threads.

Though, each framework is unique - we may suggest commonly used approaches.

Enabling EurekaLog for threading frameworks

Typically you can not control thread creation inside framework. Threads are created and managed entirely by the framework. This means that you can not use code like BeginThreadEx function 551 or TThreadEx class 559. Therefore, you can enable EurekaLog for framework's threads in two ways:
1. Automatic via options 569 (not recommended);
2. Manual via SetEurekaLogStateInThread function 570 (recommended).

Note: threads created by threading framework may be used as thread pool. This means that a single thread may serve many different tasks. A good code should preserve thread state - such as thread priority, FPU state, thread name, EurekaLog per-thread state, etc. Therefore, it is better to clean after your task - as indicated in this example 572.

Calling exception handler

Threading frameworks use custom exception processing most of the time. Again, the exact details on exception processing are specific to a particular framework. So, we can only look at some common possibilities:

a). Calling default exception handler
If your threading framework calls default exception handler to handle thread exceptions - then you don't need to do anything (since default exception handler will be hooked by EurekaLog).

b). Re-raising exception
Some frameworks may automatically capture thread exception and re-raise it into caller thread. If this is the case - then you don't need to anything (since exception from thread will be captured by exception handling code of your existing thread).

c). OnTerminate-like event
One of the typical ways to handle exceptions is utilizing OnTerminate-like event handler. A framework may offer a feature to register/install a custom callback function which will be called when thread or task is completed - so called OnTerminate event. You can register your function as handler for such event and analyze thread termination state. You can see an example of this approach in the similar illustration for the TThread class 556.

d). Explicit try/except block
For all other cases it may be easier to just wrap your thread code into explicit try/except

block and call exception handler manually. See example here 558.

Additional notes

EurekaLog has additional support for OmniThreadLibrary:

- Thread names from OmniThreadLibrary are passed to EurekaLog;
- Call stacks for exceptions in OmniThreadLibrary threads are filtered to exclude service calls (OmniThreadLibrary internal functions).

You should include EOTL unit into your application to enable the above mentioned integration. EOTL unit is located in \Source\Extras folder of EurekaLog installation.

See also:

- BeginThreadEx Function 551
- TThreadEx Class 559
- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.1.8 Thread Pools

A thread pool is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application. The thread pool is primarily used to reduce the number of application threads and provide management of the worker threads.

Delphi and C++ Builder do not have any implementations for thread pools out of the box. Applications that want to use thread pool may utilize system API (which is most commonly used in the form of QueueWorkItem function) or frameworks 565. The specifics are different for each API/framework, but there is a common scheme: you call some "create" function and pass your function as argument (a callback) - which is called a "task". This action will schedule your function (i.e. callback/task) to be executed by some thread from thread pool at the specified moment. An important moment here is that you don't know which thread will run your code. Thread for each task is selected (assigned) by thread pool manager. Primary task of thread pool is re-using threads to avoid performance penalty for creating and terminating threads. This means that single thread from thread pool will execute many different tasks. If you schedule same task multiple times - there is no guarantee that your task will be executed by the same thread from thread pool.

This means that arbitrary code is not safe for thread pooling. Task function (and all functions called from task function) must be thread-pool safe. A safe function does not assume that the thread executing it is a dedicated or persistent thread. In general, you should avoid using thread local storage or making an asynchronous call that requires a persistent thread, such as the RegNotifyChangeKeyValue function. A normal rule of thumb for thread pool tasks: do not change thread state. For example, you should not terminate thread, you should not alter FPU state, you should not alter thread priority, you should not let exceptions escape your task function, etc. So, if you need to perform non-thread-pool safe action - you need to revert it before returning control from your task function.

An example of a good thread-pool function:

```
function MyTask(lpThreadParameter: Pointer): Integer; stdcall;
begin
  try
    // ... your task code ...
  except
    // Handle any exception in task
    ApplicationHandleException(nil); // from Classes unit
  end;
  Result := 0; // Result is ignored
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  IsMultiThreaded := True;

  // Execute MyTask in a background thread
  Win32Check(QueueUserWorkItem(MyTask, nil, 0));
end;
```

Since a recommended approach 568 is to manually control activation 570 of EurekaLog per each thread - we suggest you to use such code:

```
function MyTask(lpThreadParameter: Pointer): Integer; stdcall;
begin
  // Mark thread for yourself
  NameThread('This is thread pool thread with my task');
  SetEurekaLogStateInThread(0, True);
  try
    try
      // ... your task code ...
      Result := 0; // <- to indicate success
    except
      on E: Exception do
        Result := HandleException(E); // <- to handle exception and indicate failur
    end;
  finally
    // Clean after yourself
    // (i.e. mark/prepare thread for other tasks)
    NameThread('');
    SetEurekaLogStateInThread(0, False);
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  IsMultiThreaded := True;

  // Execute MyTask in a background thread
  Win32Check(QueueUserWorkItem(MyTask, nil, 0));
end;
```

This sample code enables EurekaLog (and sets thread name) only for the duration of our task. When our task is completed, thread will be released into thread pool - so we disable EurekaLog (because we don't know which task will use this thread in the next time).

Important note: EurekaLog has to be enabled for background threads 568. And the same "thread persistent state" rule applies to EurekaLog per-thread state: you should clean after your task - as indicated in the above example.

Some thread pool implementations may have less restrictive rules. For example, thread pool implementations in Delphi/C++ Builder frameworks will probably have exception handling code established. See working with multithreading frameworks 565 for more information.

See also:

- BeginThread Function 549
- BeginThreadEx Function 551
- TThread class 553
- TThreadEx Class 559
- Multithreading frameworks 565
- Enabling EurekaLog for background threads 568
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.2 Enabling EurekaLog for background threads

EurekaLog has two states which controls if EurekaLog should be active or not: a **global** state and a **local** (per-thread) state.

1. A global state is controlled by "Activate EurekaLog" option 234 and it could be changed in run-time with `SetEurekaLogState` routine. Disabling global state will disable EurekaLog in entire application.
2. A local state is a per-thread state - i.e. each thread has its own local state which is independent from states of other threads. The local state for background threads is controlled by "Default EurekaLog state in new threads" option 246 (the main thread is always activated by default) and it could be changed in run-time with `SetEurekaLogStateInThread` routine. Disabling local state will disable EurekaLog in the current thread, but will keep it activated in other threads.

EurekaLog will be active in a particular thread only if both conditions are true:
1. Global state is enabled (`IsEurekaLogActive` is `True`).
2. Local state is enabled (`IsEurekaLogActiveInThread` is `True`).

If global state is enabled, but local per-thread state is disabled - then EurekaLog will be disabled (in this thread). If local state is enabled, but global state is disabled - then EurekaLog will be disabled (again, in this thread only).

No additional actions required for single-threaded applications (apart from enabling EurekaLog for your application): because a single-threaded application contains only main thread, and EurekaLog is always enabled for main thread by default.

Multi-threaded applications is more complex, as multi-threaded application contains additional threads. EurekaLog should be enabled in a particular thread in order to work. Main thread is always enabled - regardless of EurekaLog's options. However, any background thread is disabled by default. You have to enable EurekaLog for your background threads. This can be done in two ways:
1. Automatic 569, via options (not recommended);
2. Manual 570, via code:
   a. Using BeginThreadEx 551/TThreadEx 559 (**recommended**)
   b. Via calls to SetEurekaLogStateInThread 570.

See also:

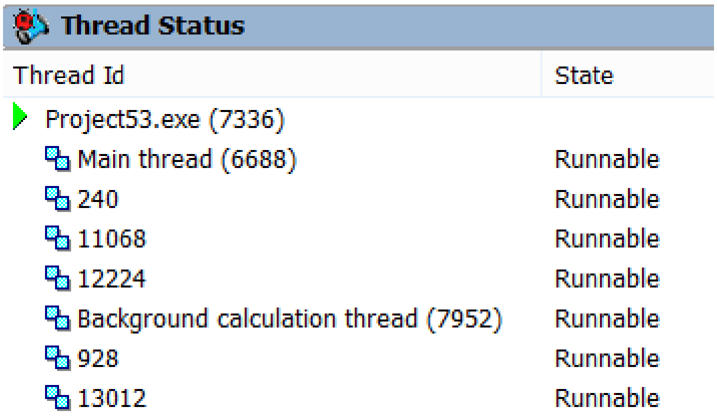
- Creating threads 547
- Multithreading options 246
- Multithreaded call stacks 85
- Configuring call stack 48

11.8.2.1 Automatic / options

Automatic way is a simple "Default EurekaLog state in new threads" option[246]. This option is set to "Disabled (recommended)" value by default. You can change this option to "Enabled" or "Enabled for RTL threads, disabled for Windows threads" states: EurekaLog will be enabled automatically when creating corresponding threads.

Important note: it is highly not recommended to use automatic enabling. Enable EurekaLog manually for each of your threads[570].

However, the problem with automatic enabling is that EurekaLog will be enabled for ALL threads in your application. Consider a typical multi-threaded application:



Thread Status

Thread Id	State
Project53.exe (7336)	
Main thread (6688)	Runnable
240	Runnable
11068	Runnable
12224	Runnable
Background calculation thread (7952)	Runnable
928	Runnable
13012	Runnable

Threads in typical multi-threaded application

This simple application has only two threads created by you (your code): a main thread (6688) and one background thread (7952). Your code created background thread to perform lengthy calculations and to avoid GUI freezing. All other threads were created by some 3rd party code:

- There are some threads servicing system thread pool[566];
- There are service threads for system background tasks (such as doing async work for WinSock);
- There are threads created by frameworks that you are using (such as Indy) - mostly to perform tasks asynchronously;
- There are even threads created by Delphi/C++ Builder IDE to perform debugger tasks!

Obviously, almost all of these threads were launched with some exception handling code in mind. Creator (3rd party code) expects these threads to behave in some specific way when exception is raised in a thread. Injecting EurekaLog processing and handling into every thread would be risky and unnecessary. Normally, you should only enable EurekaLog for threads which you know. That is threads created by you, or threads which runs your code (such as threads from thread pool). You should not enable EurekaLog for arbitrary/unknown threads.

For this reason it is highly recommended to use manually controllable activation. You can use BeginThreadEx function[551] instead of BeginThread function[549] and use TThreadEx class[559] instead of TThread class[553] to create EurekaLog-enabled threads. If you can not control thread creation code (think of thread pools[566] as example) - then you can use SetEurekaLogStateInThread function[570].

However, there still may be cases when you want to use "Default EurekaLog state in new threads" option:

- Firstly, this option remains the most simple way to enable EurekaLog for threads. You don't need to review/change your code (e.g. you don't need to search&replace "BeginThread -> BeginThreadEx" and "TThread -> TThreadEx").
- Second, there may be cases when you have no control over thread's code at all, but you still want to get bug reports for exceptions in such thread (i.e. thread's exception handling code just invokes default exception handler, which is hooked by EurekaLog).

Important Note: You can change "Default EurekaLog state in new threads" option for the above usage cases, but we recommend to use "Enabled for RTL threads, disabled for Windows threads" position when possible. Never turn "Default EurekaLog state in new threads" option into "Enabled" position until really needed.

See also:
- Manual activation 570:
  - BeginThreadEx 551
  - TThreadEx 559
  - SetEurekaLogStateInThread 570
- Multithreading options 246

11.8.2.2 Manual / code

A most simple way to enable EurekaLog in thread manually is to use BeginThreadEx 551/ TThreadEx 559:
- Use BeginThreadEx function 551 instead of BeginThread function 549: created thread will be EurekaLog-enabled (by default).
- Use TThreadEx class 559 instead of TThread class 553: created thread will be EurekaLog-enabled (by default).

Both routines are from EBase unit.

Important note: never use CreateThread function. Use BeginThread/BeginThreadEx function instead.

See the above links for more details about this recommended approach.

If you can not use BeginThreadEx/TThreadEx in your code (e.g. you do not create threads - think of thread pools 566 as example), then you have to manually enable EurekaLog for your threads, for example:

```
function ThreadFunc(Parameter: Pointer): Integer;
begin
  try
    // Both routines are from EBase unit -
    // thus both can be used in the application without EurekaLog

    // 1. Name thread for easy identification in debugger and bug reports
    NameThread('This is my thread with Parameter = ' +
      IntToHex(NativeUInt(Parameter), SizeOf(Pointer) * 2));

    // 2. Activate EurekaLog for this thread.
    SetEurekaLogStateInThread(0, True);

    // ... your normal code for the thread ...
    Result := 0;
  except
    ApplicationHandleException(nil);
    Result := 1;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  TID: Cardinal;
begin
  CloseHandle(BeginThread(nil, 0, ThreadFunc, nil, 0, TID));
end;
```

Notice first actions in the thread: setting thread name (description) and enabling EurekaLog for this thread. This template is a recommended code template. Each of your EurekaLog-enabled threads should start with these two actions. Here is the same example for TThread class:

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

procedure TMyThread.Execute;
begin
  // Service calls first:
  NameThread('This is my thread ' + ClassName);
  SetEurekaLogStateInThread(0, True);

  // ... your normal thread code ...
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TMyThread;
  E: TObject;
begin
  Thread := TMyThread.Create(False);
  try
    Thread.WaitFor;
    E := Thread.FatalException;
    if Assigned(E) then
    begin
      PPointer(@Thread.FatalException)^ := nil;
      raise E;
    end;
  finally
    FreeAndNil(Thread);
  end;
end;
```

...and for anonymous threads:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Thread: TThread;
begin
  Thread := TThread.CreateAnonymousThread(
    procedure
    begin
      // Service calls first:
      NameThread('This is my anonymous thread');
      SetEurekaLogStateInThread(0, True);

      // ... your normal thread code ...
    end);
  Thread.OnTerminate := Self.HandleExceptionInThread;
  Thread.Start;
  Thread := nil;
end;
```

...and a very similar code may be applied to any multi-threading framework that you may use: just insert calls to NameThread + SetEurekaLogStateInThread before your actual thread code.

Since a single thread may serve multiple tasks in a thread pool - it is recommended to clean after yourself:

```
function MyTask(lpThreadParameter: Pointer): Integer; stdcall;
begin
  // Mark thread for yourself
  NameThread('This is thread pool thread with my task');
  SetEurekaLogStateInThread(0, True);

  try
    // ... your task code ...
    Result := 0; // <- to indicate success
  except
    on E: Exception do
      Result := HandleException(E);
  end;

  // Clean after yourself
  // (i.e. mark/prepare thread for other tasks)
  NameThread('');
  SetEurekaLogStateInThread(0, False);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  IsMultiThreaded := True;
  Win32Check(QueueUserWorkItem(MyTask, nil, 0));
end;
```

This sample code enables EurekaLog (and sets thread name) only for the duration of our task. When our task is completed, thread will be released into thread pool - so we disable EurekaLog (because we don't know which task will use this thread in the next time).

See also:

- BeginThreadEx Function 551
- TThreadEx Class 559

- Automatic EurekaLog enabling for background threads 569
- Creating threads 547

11.9 Using Windows Error Reporting

Enter topic text here.

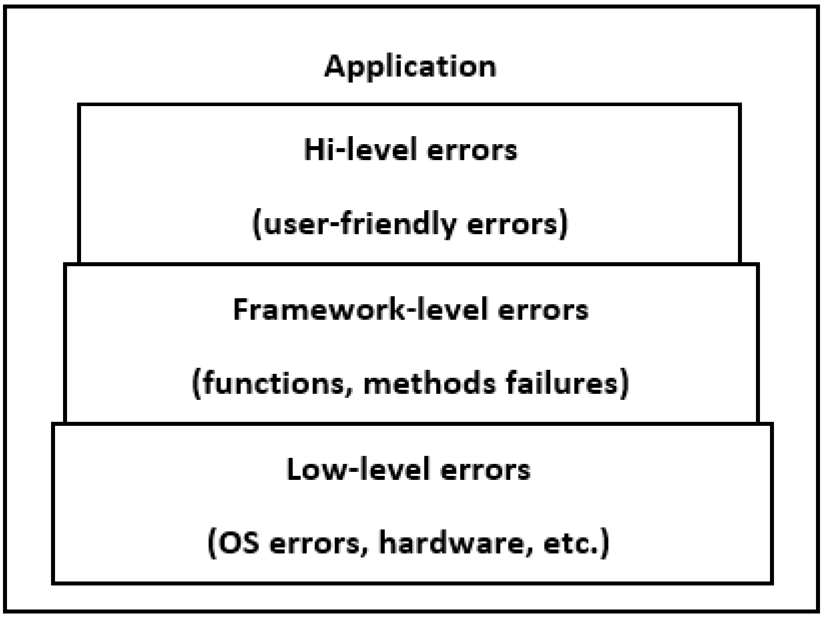
11.10 Working with minidumps

Enter topic text here.

11.11 Nested/Chained exceptions

Building good error handling facility in your applications

Application with good and thoughtful error handling will probably use layered error handling scheme.



Layered error handling architecture

That is, errors will first be raised as so-called **low-level errors**. Low-level error indicates exact reason for failure - such as failures inside OS functions, hardware exceptions (access violation, etc.) and so on. Usually, you're interested in low-level errors to get precision information about failure (so you can resolve it).

Unfortunately, low-level errors suffers from the following:
1. Low-level errors from one piece of code are not different from errors from another piece of code. I.e. they are almost identical for any code in your application (access violation from any function is represented by the same `EAccessViolation` exception class). This does not allow you to build error handling logic which can differentiate between errors.
2. Error message of low-level errors are, well, "low-level" (such as "Range Check Error", "Index out of bounds", "Access Violation at address ... in module ... read ...", etc.). Such error messages may be good for diagnostic purposes for developers, but they are not user friendly. Normal users of your application could not read them. It would be better to show more friendly messages (such as "Sorry, I can not open your file *XYZ*, it seems damaged").

Application can use **medium-** and **hi-level** errors to address these issues. Usually this means that framework should handle low-level errors and translate them into **framework-specific errors**. Framework-specific errors can include additional information about errors. In other words, low-level errors are "surfaced" to framework code and are transformed into framework exception classes. For example:

```
type
  // Declare root class for all exceptions from some framework
  EFrameworkError = class(Exception);
    // Declare subclass for all file-related errors
    EFrameworkFileError = class(EFrameworkError)
    private
      // This subclass will hold additional information: file name
      FFileName: String;
    public
      constructor Create(const AMessage, AFileName: String);
    end;
      // Declare more subclasses to specify errors even more
      EFrameworkFileCreateError = class(EFrameworkFileError);
      EFrameworkFileOpenError = class(EFrameworkFileError);
      EFrameworkFileReadError = class(EFrameworkFileError);
      EFrameworkFileWriteError = class(EFrameworkFileError);
    // ... other errors from framework
    EOtherFrameworkError = class(EFrameworkError);
    // ...

{ EFrameworkFileOpenError }

constructor EFrameworkFileError.Create(const AMessage, AFileName: String);
begin
  inherited Create(AMessage);
  FFileName := AFileName;
end;

// ------------------------------------------------------------

type
  // A sample class from framework
  TFrameworkDocument = class
  private
    FFileName: String;
    FFile: TFileStream;

  protected
    procedure Open;
    procedure ReadFileInfo;
    // ...
    procedure Close;

  public
    constructor Create(const AFileName: String);
    destructor Destroy; override;
    property FileName: String read FFileName;
  end;

resourcestring
  // Error messages for framework errors
  rsUnableToOpenFile    = 'Sorry, unable to open file: %s';
  rsErrorReadingHeader  = 'Sorry, unable to read file header from: %s';

{ TFrameworkFile }

constructor TFrameworkDocument.Create(const AFileName: String);
begin
  inherited Create;
  FFileName := AFileName;
  Open;
end;

destructor TFrameworkDocument.Destroy;
begin
  Close;
  inherited;
end;

procedure TFrameworkDocument.Open;
begin
  Close;
  try
    FFile := TFileStream.Create
     (FileName, fmOpenReadWrite or fmShareDenyWrite);
  except
    // Catch any low-level errors (such as "access denied", etc.)
    // and wrap into framework errors with supplying more information
    // (such as file name in this example)
    Exception.RaiseOuterException
      (EFrameworkFileOpenError.Create
        (Format(rsUnableToOpenFile, [FileName]), FileName));
  end;
end;

procedure TFrameworkDocument.Close;
begin
  FreeAndNil(FFile);
end;

procedure TFrameworkDocument.ReadFileInfo;
// ...
begin
  try
    FFile.Position := 0;
    FFile.ReadBuffer({ ... });
  except
    // Catch any low-level errors (such as "error reading file", etc.)
    // and wrap into framework errors with supplying more information
    // (such as file name in this example)
    Exception.RaiseOuterException(
      EFrameworkFileReadError.Create(
        Format(rsErrorReadingHeader, [FileName]), FileName));
  end;
end;
```

Note: `Exception.RaiseOuterException` construct is available only in RAD Studio 2009+. Older versions of Delphi and C++ Builder have to use "**raise**" and "**throw**" keywords.

This example illustrates a good exception handling approach for frameworks and any middle-level code. Re-raising low-level errors as framework exceptions allows you to specify more information about error: such as file name and operation kind (i.e. open, read, etc.). Such information may not be available for low-level errors. This approach also allows you to provide more descriptive error message.

In this example:

- Errors from `TFileStream` object are low-level errors. They are nested into framework exception classes.
- Errors from framework are triggered by low-level errors.

What are nested/chained exceptions

Chained exception is an exception which occurs during processing of another exception 40. That "another" (original) exception is called "**nested exception**". For example:

```
try
  // Low-level error (a.k.a.  original, first, bottom, inner, nested, root)
  raise ERangeError.Create('Invalid item index');
except
  // High-level error (a.k.a. introduced, last, top, outer, chained)
  raise EFileLoadError.CreateFmt('Error loading file %s', [FileName]);
end;
```

As you can see, low-level exception (nested) is the exception you're interested in. It indicates a reason for failure. This is what you typically want to be logged. Chained exception is triggered by original exception and provides more descriptive error message. So, you typically want to show it to user as error message.

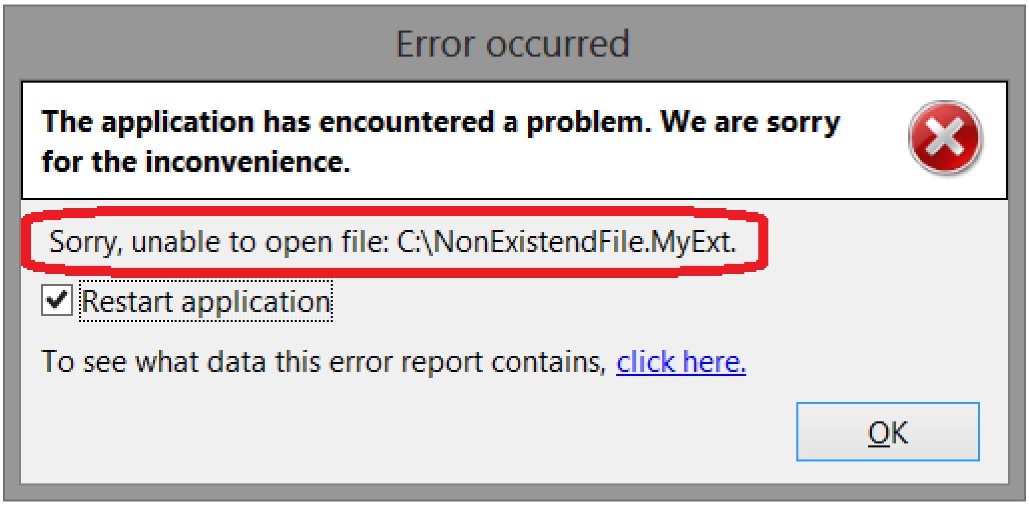
Thus, typically you want first exception to be logged, but last exception to be shown to end user. Classic/default Delphi and C++ Builder behavior is to work only with last exception always.

Delphi 2009+ only: starting with Delphi 2009 - there was new features introduced to exceptions in RTL. Support for chained exceptions was added. There are new properties BaseException and InnerException as well as special raising construct. In this model, you need to use RaiseOuterException or ThrowOuterException to preserve original exception when raising new exception. EurekaLog implements similar model with the same properties, except it doesn't require you to use special raising construct. Any exception raising automatically saves previous (original) exception in `InnerException` property. This feature available on all supported IDE versions.

EurekaLog nested/chained exception tracking feature

Default behavior of Delphi/C++ Builder: show last (i.e. chained) exception and hide original (i.e. nested) exception. This behavior is what you want for user, but it's not what you want for diagnostic purposes. EurekaLog has the feature to change/customize this behavior. Options on "Nested exceptions" 244 page allow you to customize EurekaLog behavior related to nested/chained exceptions.

Default settings for EurekaLog is to log original (nested) exception, but show chained exception to user. For example, if you run example code from above for non-existed file:



Error occurred

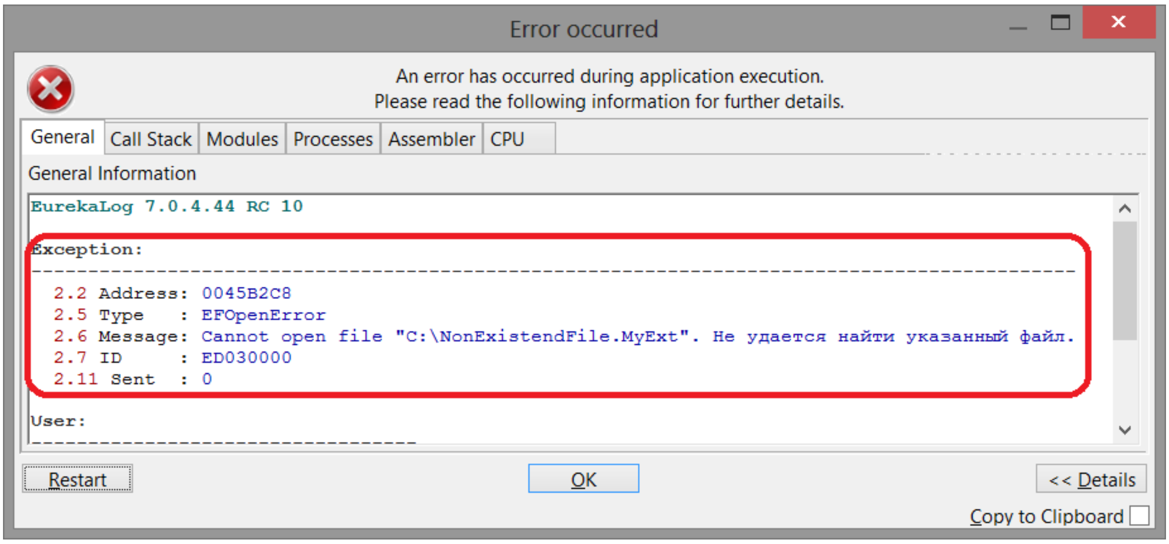
The application has encountered a problem. We are sorry for the inconvenience.

Sorry, unable to open file: C:\NonExistendFile.MyExt.

☑ Restart application

To see what data this error report contains, click here.

Error message from chained exception is shown to user
It's descriptive and user-friendly



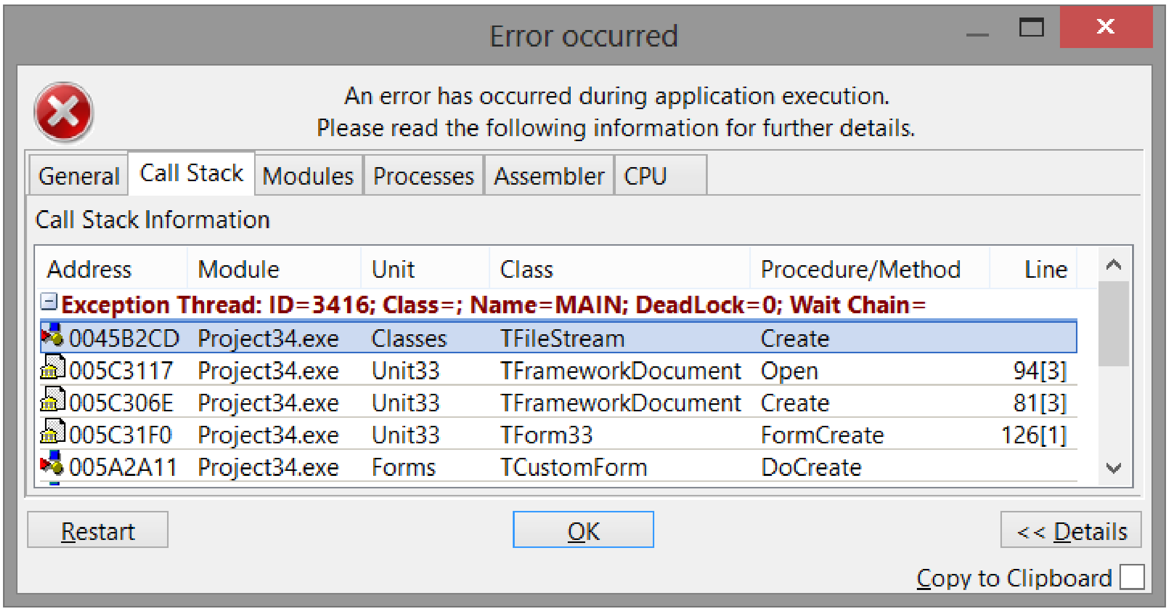
Error occurred

An error has occurred during application execution.
Please read the following information for further details.

General Information

```
EurekaLog 7.0.4.44 RC 10

Exception:
---------------------------------------------------------------------------------------------
  2.2 Address: 0045B2C8
  2.5 Type   : EFOpenError
  2.6 Message: Cannot open file "C:\NonExistendFile.MyExt". Не удается найти указанный файл.
  2.7 ID     : ED030000
  2.11 Sent  : 0

User:
-----------------------------------
```

Original (nested root) exception is stored into bug report
Its error message indicate low-level failure reason



An error has occurred during application execution.
Please read the following information for further details.

Call Stack Information

Exception Thread: ID=3416; Class=; Name=MAIN; DeadLock=0; Wait Chain=

Address	Module	Unit	Class	Procedure/Method	Line
0045B2CD	Project34.exe	Classes	TFileStream	Create	
005C3117	Project34.exe	Unit33	TFrameworkDocument	Open	94[3]
005C306E	Project34.exe	Unit33	TFrameworkDocument	Create	81[3]
005C31F0	Project34.exe	Unit33	TForm33	FormCreate	126[1]
005A2A11	Project34.exe	Forms	TCustomForm	DoCreate	

Call stack also indicate original exception

Important considerations for using nested exceptions tracking feature

EurekaLog requires ability to track life-time of exception objects for this feature. Default EurekaLog options are configured to allow it. However, if you manually changes EurekaLog options - you may disable features that allows EurekaLog to track life-time of exception objects. Such options includes:

- "Enable extended memory manager" option 250 or "Use low-level hooks" option 259 or just use Delphi 2009+
- "Capture stack only for exceptions from current module" option 237

For example, if you're using Delphi 7 and disable both "Enable extended memory manager" and "Use low-level hooks" options - then EurekaLog will be unable to detect when exception object is destroyed. Thus, tracing nested/chained exceptions feature will not function properly. This means that you may get information about wrong exception in your bug reports.

It's recommended to test your application when you alter "Enable extended memory manager", "Use low-level hooks" or "Capture stack only for exceptions from current module" options. If your application configuration fails to store proper information - please, switch nested/chained exceptions options 244 into "Classic" positions instead of (default) "Recommended" positions.

See also:

- Configuring call stack 48

11.12 Stack tracing: RAW method and frame-based method

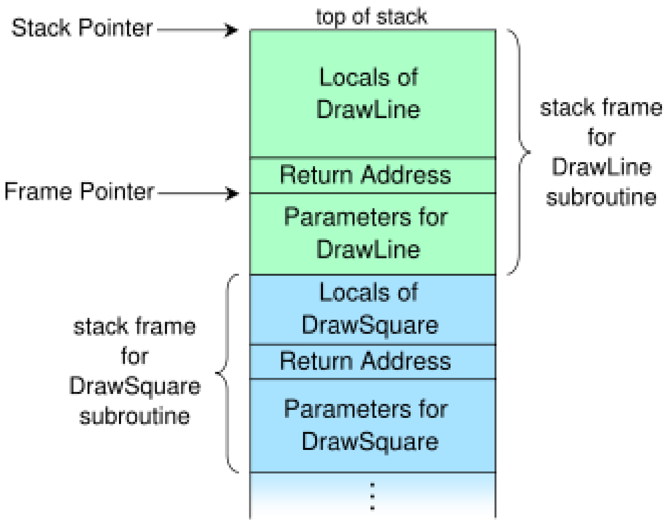
x86-32 has no reliable way to build exact call stack. All stack tracing methods use some kind of assumptions, heuristics and guessing. And sometimes tracing methods may be confused, so you may have a false-positive entries in call stack or missed entries.

x86-64 always uses frame-based stack tracing (see below), because presence of stack frames is a requirement for x86-64 calling convention. x86-64 has single stack tracing model and it's reliable until some code explicitly decides to mess with the stack.

Usually all stack tracing methods are divided into 2 groups: frame-based tracing and RAW tracing methods.

What are stack frames

x86 CPU family (both 32 bit and 64 bit) uses hardware stack to store execution path when calling routines (other CPUs - such as Itanium - use other means for that task). **Stack frames** (also called **activation records** or **activation frames**) are special data structures on stack which contains subroutine's state information. Each stack frame corresponds to a call to a subroutine which has not yet terminated with a return. For example, if a subroutine named `DrawLine` is currently running, having been called by a subroutine `DrawSquare`, the top part of the call stack might be laid out like this:



The stack frame at the top of the stack is for the currently executing routine. The stack frame usually includes at least the following items (in push order):

- the arguments (parameter values) passed to the routine (if any);
- the return address back to the routine's caller (e.g. in the DrawLine stack frame, an address into DrawSquare's code); and
- space for the local variables of the routine (if any).

A stack frame has a field to contain the previous value of the frame pointer register, the value it had while the caller was executing. For example, the stack frame of `DrawLine` would have a memory location holding the frame pointer value that `DrawSquare` uses (not shown in the diagram above). The value is saved upon entry to the subroutine and restored upon return. Having such a field in a known location in the stack frame enables code to access each frame successively underneath the currently executing routine's frame, and also allows the routine to easily restore the frame pointer to the caller's frame, just before it returns.

In other words, stack frames build a chain of frames: each frame contains link to previous frame.

It's important to understand that stack frames are not enforced on x86-32. They may be or may be not created - the decision is up to compiler. Only naked return address may be stored, without any additional information and without link to previous frame. However, stack frames always must be created for x86-64 - because this is a platform's requirement.

Note: x86-32 stack frames are different from x86-64 stack frames. x86-64 uses additional information about functions to work with stack frames. This meta-information is generated by compiler and stored inside executable.

Frame-based stack tracing

Frame-based stack tracing builds call stack by using sequence of call frames, which are added to the stack on call of most routines. Usually this method gives acceptable results (if you don't have too many very short routines - see below). You can increase it's effectiveness by enabling “Stack Frames” option. This method is quite precise and fast as it looks only for “registered” calls. It don’t scan the entire stack – just walks by sequence of frames, where every call frame points to another. This method will not work if FPO (frame pointer omission) is used (x86-32 only), in cases of non-standard stack's usage (non-standard calling convention) or in case of stack corruption.

Note: the stack tracing method for x86-64 is a frame-based stack tracing method.

RAW stack tracing

RAW tracing method works differently: it just scan the whole stack, trying to find "possible" return addresses. Really: the frames may be or may be not present in the stack (see below). But return addresses are always there. The problem is that there is no 100% way to find them. So, RAW method takes every integer in stack and tries to guess: does it look like return address? For this reason, the RAW tracing methods can be used only with some heuristic algorithm. Created call stacks may differ significantly, depending on quality of the heuristic. Present or missed debug information can also affect call stacks, because heuristic may use debug information to verify return addresses. Consider the following code:

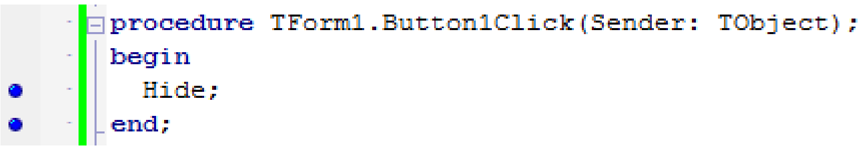
```
function LoadResString(ResStringRec: PResStringRec): string;
var
  Buffer: array [0..4095] of Char;
begin
  // ...
end;
```

This code will allocate 4 Kb of stack memory for local Buffer variable. This code does not call ZeroMemory or FillChar to initialize Buffer to contain all zeros. Therefore, Buffer will contain whatever was on stack before calling LoadResString. This "stack trash" usually includes a lot of function calls - which are not related to current execution path. RAW method will collect all these false-positive return addresses.

Note: there is no need for RAW stack tracing methods for x86-64 platform.

How stack frames are generated (x86-32 only)

Stack frame may be omitted (not created) in some cases. Consider the following code:

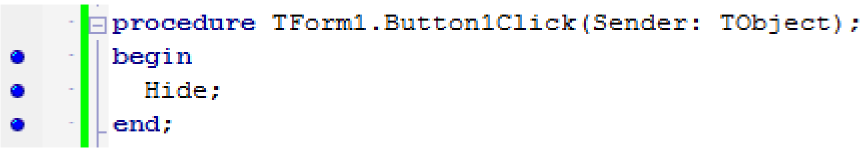


```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Hide;
end;
```

"Stack Frames" option is disabled

Notice that there is no blue "dot" on the left from "begin" line. This means that "begin" line do not generate any code. I.e. there is no code to setup stack frame in this routine. That's because this routine is very simple and do not require stack frame to manage its data.

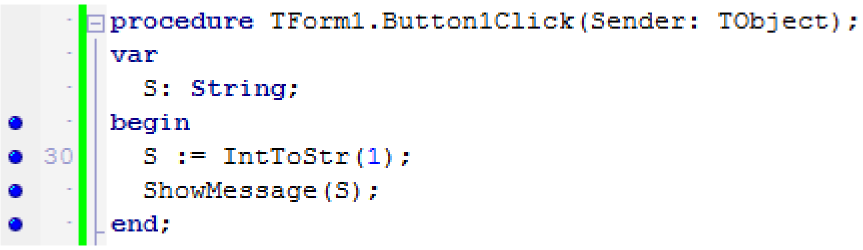
Note: "end" line generates code. This code is a simple "ret" instruction which returns control to caller.



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Hide;
end;
```

"Stack Frames" option is enabled

This is the same code, but it is now compiled with “Stack Frames” option enabled. Notice that this code has blue "dot" near "begin" line. This means that there is code to set up routine - i.e. there is code to create stack frame. "Stack Frames" option forces compiler to always create stack frames, even if there is no direct need for them.



```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: String;
begin
  S := IntToStr(1);
  ShowMessage(S);
end;
```

"Stack Frames" option is disabled

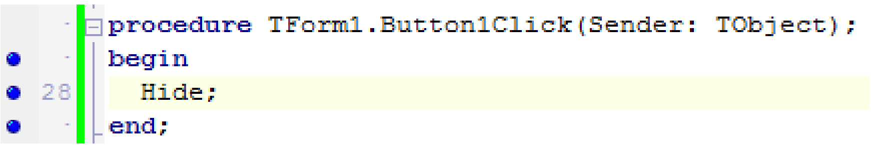
This code still have stack frame - despite fact that "Stack Frames" option is disabled. That's because this routine uses `string` type. String type is auto-managed type, so compiler need to setup initialization and finalization code and create stack frame. Any more or less complex routine (especially with local variables and arguments) will have stack frame regardless of "Stack Frames" option's state - for the same reason: compiler needs stack frame to manage routine's data.

How stack frames affects call stacks (x86-32 only)

Below are two examples on how stack frames affects call stacks.

Stack frames for offsets

EurekaLog and many other diagnostic software allow you to view so-called **offsets** 81. Line offset is calculated as a difference between the first line in routine and the line in question. For example:

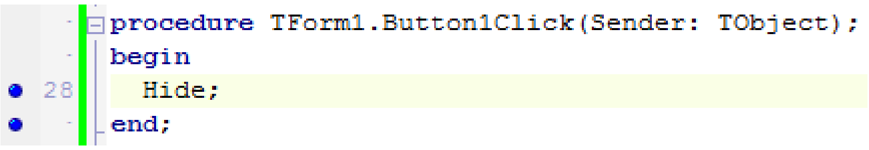


```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Hide;
end;
```

Line numbers for routine with stack frame

A call to `Hide` method is located at 28[1] line, which reads as "line #28, it has difference in one line from the first line of current routine" (some tools may indicate the same as 28[2], which reads: "second line in the routine" - in other words, some tools may use 1-based numbering instead of 0-based). Anyway, the point here is that the first line for routine is "begin" line. It's not "procedure", it's not first line of source code. That's because start of the routine is defined by code. And first code for this routine appears in "begin" line - which is indicated by a blue dot across "begin" line in editor's gutter. Therefore, "begin" line is the first line in this method (#27), "Hide" line is the second line in this routine (#28). Difference is 1.

As you may have guessed for now: line offsets are affected by stack frames. Consider the same code without stack frame:



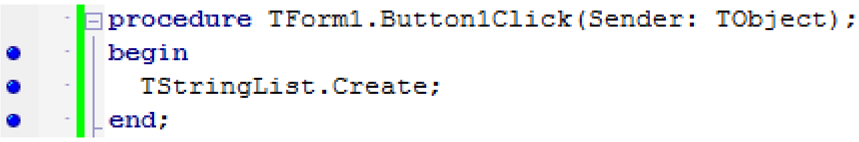
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Hide;
end;
```

Line numbers for routine without stack frame

A call to Hide method is now located at 28[0] line, which reads as "line #28, it has zero difference from the first line of current routine - i.e. it is actually a first line" (again, some tools may use 28[1] for the same fact). First code for this routine appears in "Hide" line in this case. Therefore, "begin" line is not the first line in this method, it does not generate any code, "Hide" line is the first line in this routine (#28). Difference is 0 (sinse it's the same line).

Stack frames for frame-based methods
RTL and VCL units are compiled with "Stack Frames" option turned off. This means that any frame-based method will not be able to find short routines in RTL/VCL units.

However, this fact also have less obvious consequence. Consider this code:



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TStringList.Create;
end;
```

A simple object leak with "Stack Frames" option enabled

This code creates a simple memory leak by leaking instance of `TStringList`. EurekaLog will detect a leak and build call stack for it. As you can see, `Button1Click` routine has stack frame due to "Stack Frames" option being enabled. One may expect that frame-based tracing method will discover a call to `Button1Click` and add it to the call stack.

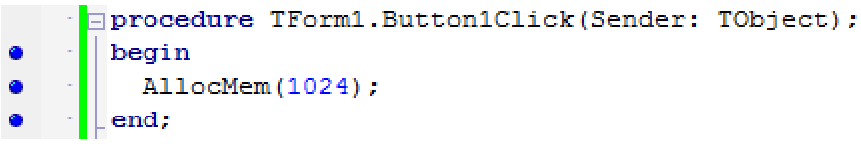
This is not so.

The stack frame for `Button1Click` allows method to identify **the caller** (in this case: `TControl.Click`). That's because stack frame does not contain information about routine itself. It contains information about the caller: return address. Return address for `Button1Click` routine will point to `TControl.Click`.

But what about `Button1Click`? Since `TStringList` is a class from RTL - it's compiled without "Stack Frames" option. Thus, constructor does not have stack frame (because it's very simple code and do not require stack frame). Therefore, it's not possible for frame-based tracing method to identify `Button1Click` routine.

Note: you can recompile RTL/VCL with "Stack Frames" option turned on.

However, if you change code like this:



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AllocMem(1024);
end;
```

Memory leak with "Stack Frames" option enabled

This code will leak anonymous memory block. `AllocMem` is routine from memory manager. It also do not have stack frame (because it's assembler stub). However, EurekaLog will be able to identify `Button1Click` routine with any stack tracing method (including frame-based one). That's because EurekaLog is awared of standard memory routines and able to extract return addresses from these routines regardless of stack frames.

The difference between two samples: first code example uses "some (unknown) code (class)", the second code example uses known memory manager function.

So, `Button1Click` routine will be detected in this case.

Conclusions

Short summary of the above discussed facts:

- There are multiple stack tracing methods for x86-32;
- There is a single stack tracing method for x86-64;
- RAW tracing method produces **more** entries in call stack, frame-based tracing method produces **less** entries (usually);
- RAW tracing method **may** show false-positive entries, frame-based tracing method **will** skip entries without proper stack frame;

- RAW tracing method is **slow**, frame-based tracing method is **fast**;
- Stack frame is indicated by **blue dot** on code editor's gutter for routine's "begin" line;
- Stack frame is created **almost always**:
  - Stack frame is created:
    - When platform is x86-64 or...
    - When "Stack frames" option is enabled or...
    - When "Optimization" option is disabled or...
    - When routine has a lot of code or...
    - When routine use auto-managed type (strings, dynamic arrays, variants, etc);
  - Stack frame is not created:
    - When platform is x86-32 and...
    - When "Stack frames" option is disabled and...
    - When "Optimization" option is enabled and...
    - When routine is too short or too simple;
- Missing stack frame will substract 1 from all line number offsets in the routine
- RTL and VCL are compiled without "Stack frames" option;
- Stack frame provides information to show entry for the **caller** in a call stack (return address);
- Stack frame for a routine does not affect if the routine in question will be shown in a call stack;
- **Immediate** caller of any memory manager functions (`AllocMem`, `GetMem`, `FreeMem`, `ReallocMem`, etc.) will be found **always**, regardless of tracing method and state of "Stack Frames" option

Important note: please remember that call stack's usefulness highly depends on available debug information from modules. EurekaLog extracts debug information with help of debug information providers 409. You have to enable (to include) required providers in case if you're going to use non-EurekaLog formats. Most remarkable formats are DLL exports 411 and Microsoft debug formats 504. Debug information providers can be enabled or disabled under Advanced / Code / Debug information 355.

See also:

- Stack tracing: deferred vs. immediate 583
- Configuring call stack 48
- Recompile RTL/VCL

11.13 Stack tracing: deferred vs. immediate

A usual solution for exception tracer is to install a callback ("hook") which is called whenever there is a new exception raised. Exception tracer creates a call stack for the exception (full bug report is not created yet). When exception is handled - bug report may be created by using pre-created call stack.

This is common approach which will work on any platform (assuming that you can hook exceptions at all). For example:

```
try
  raise Exception.Create('Example'); // <- call stack is created here
except
  Application.HandleException(nil); // <- bug report is created here
end;
```

However, it also means that each exception in your application will trigger call stack collection code. This may be a performance problem if your application uses exceptions extensively (i.e. not for reporting problems). Each raise of exception will take time to build call stack for exception. For example:

```
try
  raise Exception.Create('Example'); // <- call stack is still created here,
                     // despite the fact that bug report is not created at all
except
  on E: Exception do
    ShowMessage(E.Message);
end;
```

x86-32 platform has unique architecture: a call stack may be build later - during exception handling step. This feature can give a performance gain, because exception tracer now may build call stack later - at handling state, it's not strictly necessary to build call stack when exception is raised. Thus, there will be a huge performance boost, if most of raised exceptions are handled by your code and not reach default handler (which will create a bug report and, thus, a call stack). For example:

```
try
  raise Exception.Create('Example');
except
  Application.HandleException(nil); // <- both call stack and bug report is createc
end;

try
  raise Exception.Create('Example');
except
  on E: Exception do
    ShowMessage(E.Message);
end;
// ...no call stack is created for this case, because bug report is not created eit
```

EurekaLog 6 does not support multiple platforms. Thus, it always use optimized approach for x86-32 platform. EurekaLog 7 supports multiple platforms. Since other modern platforms do not have the similar feature, EurekaLog 7 does not implement the same logic globally. Instead, EurekaLog 7 implements universal cross-platform approach. However, EurekaLog 7 has "Delay call stack creation until handle stage" option 237. Enabling this option will bring EurekaLog 6 optimized behavior for x86-32 platform. Disabling this option will use default approach.

Thus, your Win32 applications can work with the same speed as it was with EurekaLog 6.

However, we recommend to review your code and avoid raising exceptions too often (i.e. avoid using exception as part of normal execution path; use exceptions only for errors/rare conditions) - that's because this optimization is not possible for other platforms, and your code will be slow when run on non x86-32 platform. A typical fixes for your code include:

- Avoid raising exception when you can pre-check error-condition. For example, it's better to use TryStrToInt or StrToIntDef instead of StrToInt + try/except block;
- Do not raise exception for non-errors. If you still need to do this - consider creating custom exception class for this purpose and exclude exception from EurekaLog (see next item);
- You can create custom exception classes for your purposes. You can mark some exception classes as "ignored" for EurekaLog. You can do this via filters 185, events 192 (specifically: OnRaise), attributes 190, or low-level handlers 211;
- You can also add `SetEurekaLogStateInThread(0, False)` and `SetEurekaLogStateInThread(0, True)` around blocks of code which can raise exceptions intensively, but your code handles all these exceptions.

See also:

- Stack tracing: RAW vs. frame-based 578
- Configuring call stack 48

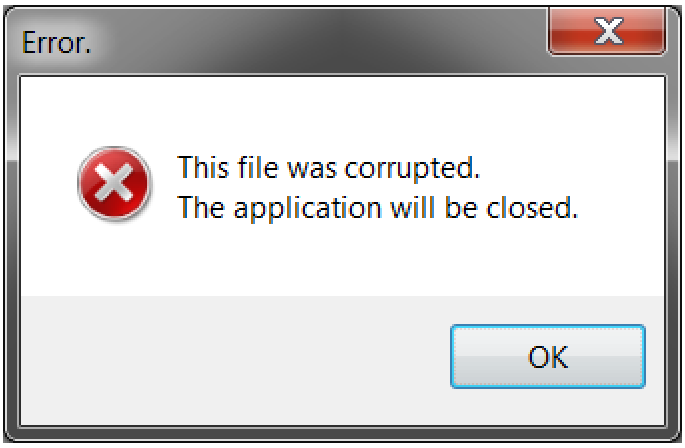
11.14 EurekaLog for shareware developers

This article will discuss EurekaLog's features that is related to shareware developers.

Checking file for corruptions

The first thing is, of course, checking the file for corruptions. You can enable this feature in "Build Options" section 349.

"**Check file corruption**" option adds check for file corruption in your project. If you enable this option, EurekaLog will calculate a CRC checksum of the compiled file and store it inside file. EurekaLog will also read this checksum from file on its startup (launch). If your executable was modified, EurekaLog will display an appropriate message and shutdown your application immediately:



Error.

This file was corrupted.
The application will be closed.

EurekaLog detected changes in executable file

You can use this option to ensure that your code wasn't modified.

Notes:

- `CheckSum` field in the `IMAGE_OPTIONAL_HEADER` structure is used to store CRC value inside executable file;
- This option checks file on disk, not running process image;
- Enabling this option will slow down loading and startup times on your executable. The bigger your executable file will be - the larger will be startup time: because the entire file must be read at startup.

As you can see it is a very simple check.

Since there is nothing that prevents the cracker from changing the CRC in the header of the file once it has been changed, then this mechanism could not be considered as significant protection. Its purpose is checking file on **unpremeditated** changes. It should not be considered as serious line of defence against crackers. If you want to protect your file from crackers (not from simple modifications) then you need to deploy your own custom protection.

See also: using EurekaLog with exe packers/cryptors/protectors 520.

Another trick here is using executable compressor or signing the application. You see, once you have feed your exe to file compressor (such as UPX, for example) or sign it with digital signature - your file is changed, therefore EurekaLog fails while checking your CRC at start-up and shutdowns your application. From your point of view: your application just stopped working after compressing (or signing). The important point there is that compressing or signing your application already implies file checking. Since compressed file is far more sensitive to changes - therefore any modern file compressor contains some sort of checking mechanism that detects changes in the very same way that EurekaLog does. The same thing is applied to file signatures. If file is signed, then OS loader (and not the application itself) will check it before execution to ensure that its signature was not broken. The conclusion here is simple: if you use file compressor or you sign your application, then you can safely turn off the EurekaLog's "Check file corruption" option.

BTW, if you are writing an installer on your own - then you probably don't want to build it as single exe-file AND enable some sort of checking on it (EurekaLog's check, file compressor or digital signature). Why? Well, imagine that you got a 200 Mb installer in one single exe file and your client launches it. Whatever mechanism you use - it will scan the ENTIRE file at start-up. And scanning 200 Mb takes a lot of time! User can think that your application hangs. It is a lot better to make a (minimum) two-files distribution (small installer exe and large application archive) or turn off protection for installer.

And, finally: remember, that even digital signature is a helper anti-malware mechanism for end-users and not an anti-crack protection! Digital signature on exe can only ensure user that this file was not modified and don't come from hacker instead of real application developer. It does not protect your application from cracker, since cracker can just simply remove digital signature.

Debug information: Pros and Cons

Another very important feature for shareware developers is a debug information 40. EurekaLog embeds debug information into your executable 38 to help you track source of your bugs. By default debug information is included only in dcu-files and don't go into your exe file. It is not possible to display a proper call stacks with methods and units names and source lines without the debug information. Because of this, EurekaLog gather all debug information from your application and attaches it to executable as a resource.

The debug information is not stored in clear text, of course - it is compressed and encrypted. But since the password is stored in the exe itself - again, nothing prevents cracker from reading the password and decoding debug information. What kind of bonus can receive a cracker from debug information? Well, debug information contains names of units, classes, methods, function and procedures. Plus line numbers. It does not contain your source code, but, indeed, it is very interesting information.

Because of it, the very common question is: "isn't an executable file compiled with EurekaLog easier to crack?".

Well, it depends on many settings.

Let's check it out.

Yes, it is true, that including debug information to your application CAN make cracker's life easier. But it do not mean, that crack will appear automatically. Cracker still need to study your protection, and if your protection is written good, then it is not contained in dedicated procedures, but rather spread over all of your code. Since there is no "protection procedures", then reading procedure names gives no valuable information (and if you still have those - just give them a non-descriptive names). Surely, you can exclude certain "sensitive" routines from debug information by using `{$D-}` before routine and `{$D+}` after routine, but this will introduce gaps in debug information, which could be noticed by a cracker.

In any case, even if debug information included in clear text (which is not, even without password), it can only speed up the study of your application. It can not suggest a way to crack. With or without debug info cracker will do the very same things. Therefore the difference "debug info" <> "no debug info" is equivalent of "crack can appear sooner" <> "crack can appear later", but not the equivalent of "app is cracked" <> "app can not be cracked".

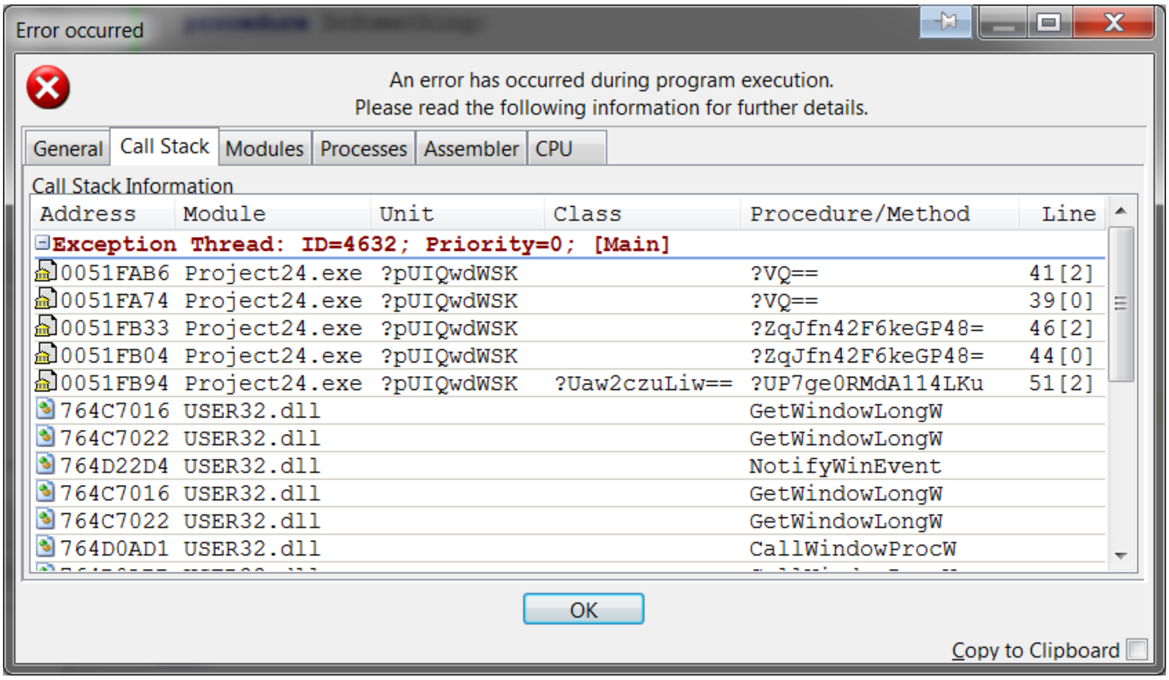
Though you can do a certain steps to prevent cracker from gaining even this speed-up bonus from your debug information.

First: you can exclude certain procedures from being included into debug information attached to exe. Just wrap them in compiler's directives that switches debug information on and off (it's `{$D+/-}`). But, again, the cracker can see the "white spaces" in debug info coverage (if that idea comes to his mind too), so this will lead him directly to your protection code (good for honeypots though).

Second: you can disable adding methods, functions and procedure names to your executable. You can do it by enabling "Do not store the Class/Procedure names" option 243. With enabled option the only information that goes into your exe file is units names and line numbers. Of course, now your reports do not contains these names too. But call stack is still useful to you, since there are unit names and line numbers with offsets - these are enough to track down the problem, though it may be harder to do.

See this article 578 for detailed explanations of the offsets.

Third: if you *really* worry about your names - you can encrypt debug information with custom password, which isn't stored in executable. You can do it by typing password in the "Debug information encryption password" option 243. So debug information will not be useful without a password (which is not stored in application).



An error has occurred during program execution.
Please read the following information for further details.

Call Stack Information

Address	Module	Unit	Class	Procedure/Method	Line
Exception Thread: ID=4632; Priority=0; [Main]					
0051FAB6	Project24.exe	?pUIQwdWSK		?VQ==	41[2]
0051FA74	Project24.exe	?pUIQwdWSK		?VQ==	39[0]
0051FB33	Project24.exe	?pUIQwdWSK		?ZqJfn42F6keGP48=	46[2]
0051FB04	Project24.exe	?pUIQwdWSK		?ZqJfn42F6keGP48=	44[0]
0051FB94	Project24.exe	?pUIQwdWSK	?Uaw2czuLiw==	?UP7ge0RMdA114LKu	51[2]
764C7016	USER32.dll			GetWindowLongW	
764C7022	USER32.dll			GetWindowLongW	
764D22D4	USER32.dll			NotifyWinEvent	
764C7016	USER32.dll			GetWindowLongW	
764C7022	USER32.dll			GetWindowLongW	
764D0AD1	USER32.dll			CallWindowProcW	

Encrypted call stack

Well, you can not view the call stack (and assembler information) directly - it will be encrypted (EurekaLog uses TEA - this well-known cipher has very compact code). But you can load such encrypted report into EurekaLog's Viewer 617, enter your password (which you do know and cracker is not) and view unencrypted report.

How reliable is this method? TEA has few weaknesses. TEA is susceptible to a related-key attack which requires chosen plaintexts. Because EurekaLog projects is compiled by you and it happens rarely (much less that about 8'000'000 projects which is required for this attack to succeed) - this attack type is not applicable to you. TEA also suffers from equivalent keys, which means that the effective key size is 126 bits instead of 128 bits.

This means that cracker is left with brute-forcing 126 bit key or dictionary attack (which you can avoid just by using random passwords). Therefore this protection method is quite reliable, and you shouldn't be worried about your routine names to be accessible to everyone.

See also:

- Using EurekaLog with exe packers/cryptors/protectors 520
- Configuring call stack 48

11.15 What's the difference between SSL and TLS send modes?

Some send modes have options to turn on SSL and TLS mode for sending. What's the difference?

Basically, SSL mode and TLS mode are just short **labels** to mark the following behaviour:

SSL mode

Early implementations of encrypted protocols used a different TCP port from normal protocol, and expected an encryption negotiation to start **immediately**, instead of waiting for a special command from the client using the standard port. Such protocol is usually called by adding "s" before or after protocol's name. For example: "FTPS" (or "implicit FTPS", not to be confused with "SFTP"), "HTTPS", "SMTPS" (or "SSMTP"). SMTP uses port 465 for this purpose, FTP uses 990 and HTTP uses 443.

Short summary:
- Encrypted connection is established immediately.
- It's implicit mode.
- Also called as: pure SSL, implicit SSL, FTPS, FTP/SSL, implicit FTPS, HTTPS, SMTPS, SSMTP, implicit SMTP, secure SMTP.
- Ports: SMTP - 465 (called "SMTPS"), FTP - 990 (called "implicit FTPS"), HTTP - 443 (called "HTTPS").

TLS mode

Later implementations of protocols used a different approach. The connection is initially established to unsecured port as with normal protocol. Once a connection is established, the client issues a special command (usually it's a STARTTLS, AUTH SSL or AUTH TLS). If the server accepts this, the client and the server negotiate an encryption mechanism. If the negotiation succeeds, the data that subsequently passes between them is encrypted. Because connection is established as unsecure - the same port (compared to normal protocol) can be used. However, sometimes a different port can be used. Protocol is called "FTPES" (or "explicit FTPS") for FTP, "SMTP AUTH" or "ESMTP" (extended SMTP) for SMTP.

Short summary:
- Connection established as unsecured (plain) and switched to secure mode on demand (special command).
- It's explicit mode.
- Also called as: explicit SSL, STARTTLS, AUTH TLS, FTPS, FTP/SSL, FTPES, explicit FTP, ESTMP, SMTP AUTH, explicit SMTP.
- Ports: SMTP - 25 or 587, FTP - 21, HTTP - not applicable.

Note: such names (with "SSL" and "TLS") may be a little confusing, because both SSL and TLS are application-layer cryptographic protocols. TLS is just a successor of SSL, i.e. TLS 1.0 is SSL 3.1. TLS is application protocol independent. Higher-level protocols can layer on top of the TLS protocol transparently. The TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left to the judgment of the designers and implementors of protocols that run on top of TLS.

So, both so-called "SSL mode" and "TLS mode" will use TLS **or** SSL protocol (depending on handshake's result) for handling encrypted connections. Words "SSL" and "TLS" are used just as short convenient "*labels*" for modes, meaning "**encrypted protocol, old version, implicit**" for "**SSL**" and "**encrypted protocol, new version, explicit**" for "**TLS**".

This is common interpretation to be seen in other software as well.

You can know which mode to select by reading help/FAQ for your server. Usually, it's TLS mode, when available. If you can't figure out the proper mode - try TLS mode first. If it works - keep it, if not - switch to SSL mode.

TLS mode will revert back to plain mode, if issuing special command will be unsuccessful (for example, if server doesn't support secure mode). Such "probing" is not possible for SSL mode, since you must connect in already defined state (secured or unsecured), while TLS mode decides this while negotiating with server. That's why you may keep TLS mode always enabled (unless you need SSL mode only).

Note: EurekaLog doesn't support SFTP protocol (which is based on SSH; SFTP stands for "SSH File Transfer Protocol").

See also:

- Send modes 390

11.16 Memory leaks detection limitations

The EurekaLog memory leaks detection has some limits, which derives from its internal structure.

EurekaLog detects the memory leaks at the application's exit, so at this state the program has just freed all its non-static resources (resources allocated at run-time as objects created with the Create constructor, variables allocated with the GetMem function, etc).

This technique generating the following limitations:

1. Enabling memory leak detection has little performance penalty (no more than about 5% for memory operations), because EurekaLog needs to build call stack for each memory allocation;
2. Enabling memory leak detection has little memory usage penalty (about 200 bytes for each memory allocation on x86-32 and about 400 bytes on x86-64);
3. Memory leak report automatically hides Assembler and CPU sections;
4. EurekaLog will not execute standard events during processing memory leak reports (this is because required resources was freed);
5. Call stack for memory leak is limited to 35 entries;
6. Memory leaks detection works only for standard Delphi's heap / memory manager (i.e. GetMem/FreeMem/ReallocMem); it can not find leaks with other memory managers (like HeapAlloc/HeapFree or VirtualAlloc/VirtualFree). Use resource leaks 255 ability for that.
7. EurekaLog is unable to show human-readable call stack if DLL which created leak has been unloaded (this limitation is applicable only for applications with installed shared memory manager).
8. If you use shared memory manager - you must compile all modules with same memory manager settings. If you don't use shared memory manager - there is no additional limitations.
9. (C++ Builder only) "Dynamic RTL" is not supported in C++ Builder. If you enable "Dynamic RTL" option - EurekaLog's memory features will be disabled.

See also:

- Enabling memory features for C++ Builder 255
- Memory leaks detection options 250
- Solving memory leaks 166
- Configuring project for leaks detection 508

11.17 Resource leaks detection limitations

The EurekaLog resource leaks detection has some limits, which derives from its internal structure.

EurekaLog detects the resource leaks at the application's exit, so at this state the program has just freed all its non-static resources (resources allocated at run-time).

This technique generating the following limitations:

1. Enabling resource leak detection has little performance penalty (no more than about 5% for resource operations), because EurekaLog needs to build call stack for each resource allocation;
2. Enabling resource leak detection has little memory usage penalty (about 200 bytes for each resource allocation on x86-32 and about 400 bytes on x86-64);
3. Resource leak report automatically hides Assembler and CPU sections;
4. EurekaLog will not execute standard events during processing resource leak reports (this is because required resources was freed);
5. Call stack for resource leak is limited to 35 entries;
6. EurekaLog is unable to show human-readable call stack if DLL which created leak has been unloaded.
7. Resource monitoring is performed by hooking imported DLL functions.
8. Resource leaks are checked only for supported functions 590. Leaks in other functions will be not traced.

See also:

- Enabling memory features for C++ Builder 255
- Memory leaks detection options 250
- Solving memory leaks 166
- Configuring project for leaks detection 508

11.17.1 List of functions

This is a list of functions which are monitored by resource leaks feature. This list is expected to be expanded as we would test more functions.

Currently monitored by default functions are:

- HeapCreate/HeapDestroy
- HeapAlloc/HeapReAlloc/HeapFree
- HeapLock/HeapUnlock
- CreateCompatibleDC/CreateDC/CreateIC/DeleteDC
- GetDC/ReleaseDC

You can also add functions manually by using HookWin32API function from EResLeaks unit.

For example:

```
uses
  EResLeaks;

initialization
  HookWin32API('Kernel32.dll', 'HeapCreate',  'Heaps', 'Heaps', 3, True,  ctEqual,
  HookWin32API('Kernel32.dll', 'HeapDestroy', 'Heaps', 'Heaps', 1, False, ctEqual,

  HookWin32API('Gdi32.dll',  'CreateCompatibleDC', 'Device Contexts', 'Device Conte
  HookWin32API('Gdi32.dll',  'CreateDC',           'Device Contexts', 'Device Conte
  HookWin32API('Gdi32.dll',  'CreateIC',           'Device Contexts', 'Device Conte
  HookWin32API('Gdi32.dll',  'DeleteDC',           'Device Contexts', 'Device Conte

  HookWin32API('AdvAPI32.dll', 'CreateRestrictedToken', 'Handles', 'Tokens', 9, Tru
  HookWin32API('AdvAPI32.dll', 'OpenProcessToken',      'Handles', 'Tokens', 3, Tru
  HookWin32API('AdvAPI32.dll', 'OpenThreadToken',       'Handles', 'Tokens', 4, Tru
  HookWin32API('AdvAPI32.dll', 'DuplicateToken',        'Handles', 'Tokens', 3, Tru
  HookWin32API('AdvAPI32.dll', 'DuplicateTokenEx',      'Handles', 'Tokens', 6, Tru

  HookWin32API('Kernel32.dll', 'CloseHandle', 'Handles', 'Handles', 1, False, ctEqu
end.
```

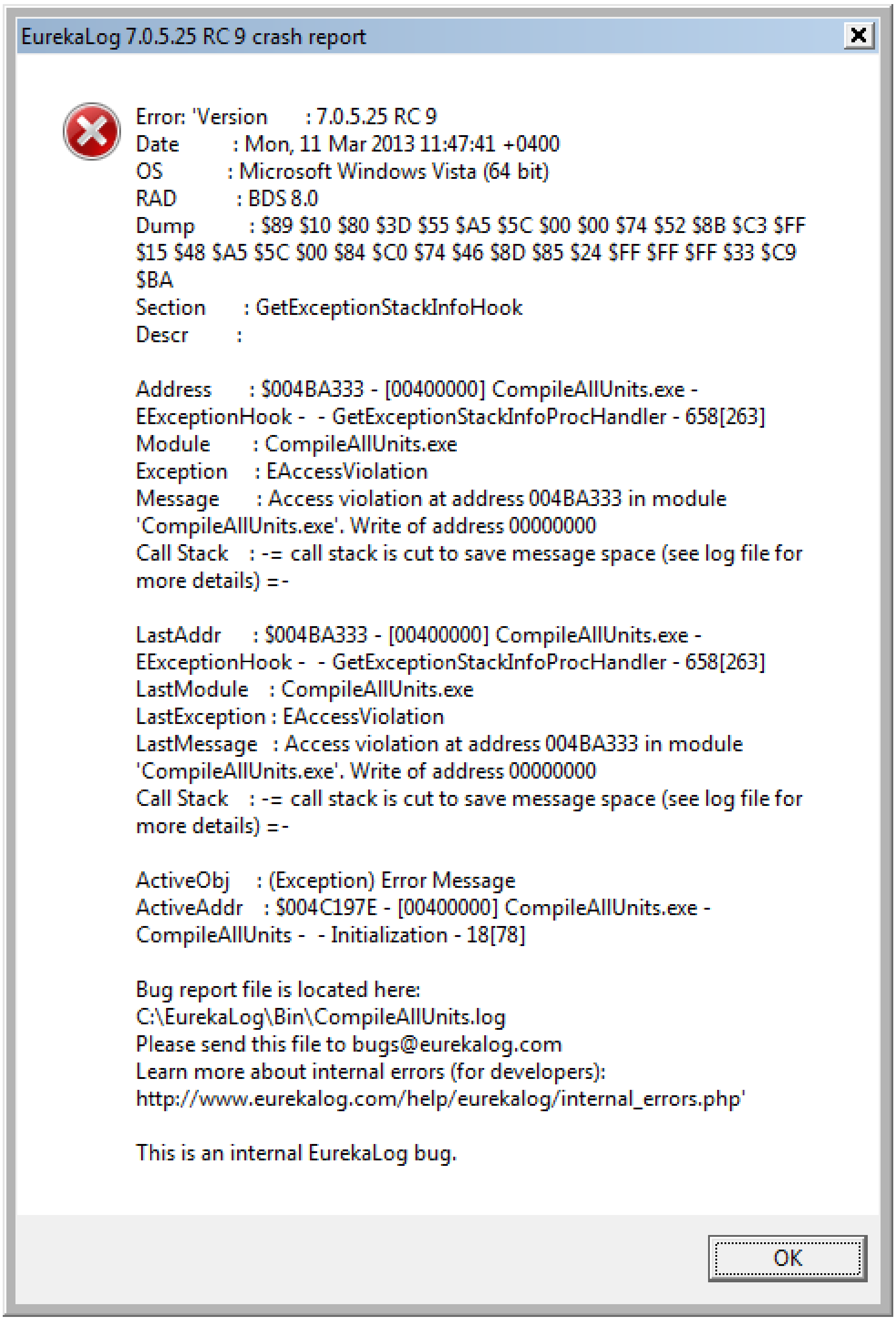
See also:

- Resource leaks options 255
- Resource leaks detection limitations 255

11.18 Internal Errors

This article is for application developers, which use EurekaLog exception tracer 40 tool. **If you are not a developer of the application - please, contact developers of application and let them know about this issue.**

Sometimes your application may show "EurekaLog crash report", also known as "Internal Error bug report". This article explains such reports.



```
Error: 'Version      : 7.0.5.25 RC 9
Date         : Mon, 11 Mar 2013 11:47:41 +0400
OS           : Microsoft Windows Vista (64 bit)
RAD          : BDS 8.0
Dump         : $89 $10 $80 $3D $55 $A5 $5C $00 $00 $74 $52 $8B $C3 $FF
$15 $48 $A5 $5C $00 $84 $C0 $74 $46 $8D $85 $24 $FF $FF $FF $33 $C9
$BA
Section      : GetExceptionStackInfoHook
Descr        :

Address      : $004BA333 - [00400000] CompileAllUnits.exe -
EExceptionHook -  - GetExceptionStackInfoProcHandler - 658[263]
Module       : CompileAllUnits.exe
Exception    : EAccessViolation
Message      : Access violation at address 004BA333 in module
'CompileAllUnits.exe'. Write of address 00000000
Call Stack   : -= call stack is cut to save message space (see log file for
more details) =-

LastAddr     : $004BA333 - [00400000] CompileAllUnits.exe -
EExceptionHook -  - GetExceptionStackInfoProcHandler - 658[263]
LastModule   : CompileAllUnits.exe
LastException : EAccessViolation
LastMessage  : Access violation at address 004BA333 in module
'CompileAllUnits.exe'. Write of address 00000000
Call Stack   : -= call stack is cut to save message space (see log file for
more details) =-

ActiveObj    : (Exception) Error Message
ActiveAddr   : $004C197E - [00400000] CompileAllUnits.exe -
CompileAllUnits -  - Initialization - 18[78]

Bug report file is located here:
C:\EurekaLog\Bin\CompileAllUnits.log
Please send this file to bugs@eurekalog.com
Learn more about internal errors (for developers):
http://www.eurekalog.com/help/eurekalog/internal_errors.php'

This is an internal EurekaLog bug.
```

A typical internal error report in EurekaLog

What are internal errors

Internal error is (fatal) exception raised during capturing information about another exception (non-fatal). This exception is unexpected and it indicates wrong state of your application. When such exception is encountered - EurekaLog stops application and shows internal error report. It's not possible to continue normal work after encountering fatal exception. Internal error report is generated to avoid application's hang or crash (indeed: exception -> capture information about exception -> get another exception -> capture information about exception -> get yet another exception -> etc. until hang or crash). It's not possible to perform normal exception processing (collect call stack, save bug report file, send bug report to developers) for the same reasons.

How to read internal error report

Original (non-fatal) exception is shown last (as Active*XYZ*):

```
ActiveObj     : (Exception) Error Message
ActiveAddr     : $004C197E - [00400000] CompileAllUnits.exe - CompileAllUnits -   -
Initialization - 18[78]
```

Unexpected (fatal) exception is shown first:

```
Address        : $004BA333 - [00400000] CompileAllUnits.exe - EExceptionHook -   -
GetExceptionStackInfoProcHandler - 658[263]
Module        : CompileAllUnits.exe
Exception     : EAccessViolation
Message                  :   Access   violation   at   address   004BA333   in   module
'CompileAllUnits.exe'. Write of address 00000000
```

Location where fatal exception was discovered is indicated by "Section":

```
Section       : GetExceptionStackInfoHook
```

Middle part of internal error report (Last*XYZ*) shows value of `ExceptionManager.LastThreadException`. Usually this part doesn't hold any additional valuable information.

Unfortunately, it's not possible to create proper call stack for internal error report: because collecting call stack means calling the crashed code again (so it's highly likely that it will cause another crash). That's why call stack show only location, which has detected the problem (up to `GetExceptionStackInfoHook` in the above example), not the location which has actually encountered a problem (there is no info in call stack about routines called from `GetExceptionStackInfoHook`).

What causes internal errors

Internal errors can be caused either by bugs in EurekaLog's code or by bugs in your code. Usually, fatal exception is access violation exception occurred when EurekaLog collects information about ordinal exception.

Internal errors can be caused by the following reasons:

- Allowing application to continue to work after it encounters its first unhandled exception. Unknown exception (especially access violation) puts your application into unknown state. This state may be unstable, so any further work may cause crashes. For example, you may get access violation exception due to memory corruption bug in your code. And you continue to run your application after handling this exception. However, memory state is still corrupted. Thus, when next exception occurs, and EurekaLog tries to work with it - EurekaLog may trigger access violation exception due to corrupted memory.
- Bugs in EurekaLog code.
- Bugs in your code.

What is panic mode

When EurekaLog encounters unexpected exception during processing of another exception - it puts itself into so-called "panic mode". This is "safe-fallback" mode designed to continue work to complete creation of internal error report.

Panic mode:

- Outputs debug messages to indicate failure condition;
- Notifies attached debugger (if present) about failure condition by triggering forced software breakpoint. If you're running application under debugger - you can stop on this breakpoint and investigate the crash;
- Disables EurekaLog in all threads. This is done to avoid calling crashed code for any possible further exceptions;

- Installs memory manager replacement. This is done to allow allocating and disposing memory blocks when internal error was caused by corrupted memory manager state.

Panic mode is used for all fatal exceptions, not just for internal errors. For example, when EurekaLog discovers corruption of memory block with its debugging memory manager - it puts itself into panic mode to allow to continue work despite memory corruption.

How to solve internal errors

Unfortunately, it's not possible to get exact reason for internal error from internal error report. That's because internal error report is created when fatal exception is discovered, not when it occurs. So, report doesn't contain information about exact location of the problem, only indication to generic code's section. Internal error report allows you to know that something is wrong in your application, but you need to do additional work to figure out reason.

If you have reproducible test case to invoke internal error - please, contact EurekaLog support to get assist with resolving your issue.

Otherwise, you may try the following:

- Configure your project for maximum debugging 58. Most notably (but not suffice): enable "Use Debug DCUs" option and remove any `{$D-}` directives in your code;
- Try to run your application under debugger. Do not disable stopping on exceptions and non-user breakpoints. You can try to investigate reason or capture screenshots to submit them later to EurekaLog's support;
- Try to enable "Enable extended memory manager" option 250;
- Try to disable "Enable extended memory manager" option 250 and use 3rd party debugging memory manager (such as FastMM in full debug mode or SafeMM);
- Try to comment your customization code 189. Exceptions raised by your event handlers sometimes may lead to internal errors;
- Try to set `_AllowBypassInternalErrors` global variable (`EBase` unit) to `True` and set `_InternalError` global variable to `nil` (do this in any place before internal error occurs). See if this can give you more information in internal error report. These options will disable creation of internal error reports and will allow you to continue your application's work on your own risk;
- Try to set `_ForceEurekaLogForInternalErrors` global variable (`EExceptionHook` unit) to `True` (do this in any place before internal error occurs). See if this can give you more information in internal error report. This option will enable EurekaLog for fatal exceptions (as mentioned above: EurekaLog is disabled by default for fatal exceptions). Of course, internal error reports must be enabled;
- Try to use internal debug mode 613.

How to change default behavior of internal errors handler

EurekaLog allows you to customize default behavior for internal errors via low-level customizations 211. You can use the following low-level possibilities:

- `EBase._InternalError` - allows you to install your own internal error handler. You can use this to show another message, to create another bug report, etc. You can also to reset this handler to `nil`.
- `EBase._AllowBypassInternalErrors` - allows you to ignore internal errors. This option has no effect if `_InternalError` handler is assigned.
- `EExceptionHook._ForceEurekaLogForInternalErrors` - allows you to use EurekaLog for internal errors. This possibly may collect more information about fatal exception, but also may cause application's crash without creating internal error report.

See also:

- Troubleshooting problems at run-time 613
- Configuring project for debugging 58
- Low-level handlers 211

Part XII

12 Troubleshooting

This section should help you to resolve problems with EurekaLog.

Problems?

- Troubleshooter
- Installation issues 596
- Activation issues 606
- Frequently Asked Questions 215
- Knowledge base
- Other problems 615

12.2 Installation problems

Note: these are troubleshooting guidelines **for EurekaLog 7 only**. If you install other version of EurekaLog - use corresponding help resources.

Please, use our installation troubleshooter first. If it did not help or you can't use it:

You should do the following to make a *clean reinstall* in case of any problems:

1. **Uninstall** all installed EurekaLog versions (see also: full manual uninstallation 605 for uninstalling EurekaLog 7).
2. **Delete all dcu files**, related to your project (you've already should have deleted dcu files related to EurekaLog on previous step).
3. **Download the latest EurekaLog version** (or RC version, if you like).
4. **Install EurekaLog under the *same user account***. I.e. if you run/work in Delphi under "User1" account - then run installer under "User1" account, not "User2".
5. Note, that there should be **your Delphi version present** during installation process. I.e. if you have Delphi 7 and Delphi 2009 installed, the EurekaLog's installer should show you both options (switches) during installation (like IDE support/Delphi/Delphi 7 and IDE support/Delphi/Delphi 2009).
6. **Restart your PC**.

Most common problems:

Q: I do not see my IDE version in the installer
A: First, make sure that you have EurekaLog for Delphi/C++ Builder (because we also have EurekaLog for .NET, which is a different thing). If you do not see all installed Delphi versions in EurekaLog's installer - then probably your Delphi installation was somehow corrupted and EurekaLog installer doesn't see your installations. Try to reinstall or repair your Delphi installation.

EurekaLog installer looks for path values in the registry. For example: `HKEY_LOCAL_MACHINE\SOFTWARE\Embarcadero\BDS\8.0\RootDir=C:\Program Files (x86)\Embarcadero\RAD Studio\8.0\` (don't forget about `Wow6432Node` node on x86-64 machines).

An easiest way to fix this is to run "Repair" or "Reinstall" of your IDE.

Q: I do not see EurekaLog menu items in IDE after installation
A: If you do not see EurekaLog in Delphi's IDE after installation (and you DID check appropriate switch during installation) - check your package settings:

1. **Automatically**. Use Start Menu / Programs / EurekaLog 7 / Manage menu item (run it under administrator account). Click on "EurekaLog 7 with IDE expert" button under your IDE to install EurekaLog.
2. **Manually**. Open IDE and go to "Component"\"Install Packages". See if there is EurekaLog package and it has a checkbox marked. If checkbox is unchecked - then switch it on. If there is no EurekaLog's expert package - then click on "Add" button and pick a package from your \bin\ folder. More information 598.

Q: I see "File not found: *XYZ*" error when I compile my project
A: EurekaLog is not registered in your IDE or it is not registered properly. Check your

search paths:

1. **Automatically**. Use Start Menu / Programs / EurekaLog 7 / Manage menu item (run it under your account). Click on "EurekaLog 7 with IDE expert" button under your IDE to register EurekaLog.
2. **Manually**. Open IDE and go to "Tools"\"Options". Set library search paths to appropriate folder in \Lib subfolder of EurekaLog installation. More information 598.

Q: I see "Unit *XYZ* was compiled with different version of *ABC*" or "Could not compile used unit *XYZ*" errors when I compile my project
A: Your project uses old or wrong files. Clean .dcu/.obj files of your project and make a full build. If this will not help - please check search paths of your project and your IDE as explained here 598.

Q: My IDE (with many installed components and extensions) hangs or crash during loading of EurekaLog or during project compilation, or it is unable to compile projects (errors like "File not found" while search paths are set)
A: Delphi and C++ Builder IDEs are 32-bit processes, which are limited to 2 Gb of address space. There are also other known limits (such as command-line length, environment variables length and search paths length). Probably you have too many components and extensions installed, so when you install or use EurekaLog - these limits are reached. Try to remove unused components and extensions, try to remove search paths for unused components or re-arrange them to move EurekaLog first. If you can not remove any component or extension - then you can use EurekaLog without using EurekaLog IDE expert. Please read this article 221 fore more information.

Note: please note that limitations of 32-bit processes do not depend on your installed hardware memory (RAM) and disk space.

Q: My IDE crashes when loading or using EurekaLog, but I do not have many components/extensions installed - only few.
A: EurekaLog installs hook in IDE to catch errors in EurekaLog itself. In some rare cases this feature may conflict with other similar hooks. Most typical examples of 3rd party software with same functionality are AQTime, madExcept and JEDI Code Library (JCL). To disable EurekaLog reporting - please uncheck "Catch EurekaLog IDE Expert errors" option 230.

Other issues:

If installer gives you some error, then make sure that anti-virus or any other scanner does not interfere with unpacking files (for example, anti-virus can block access to files due to false-positive alert). Try to clear your `%TEMP%` folder. Check if you have enough free disk space. You should also try to run installer under administrator account (run it elevated under Vista or later Windows versions). And, finally, install all available Windows updates via Windows Update.

After installation: check if there are appropriate subfolders in EurekaLog installation directory. For example, if you have Delphi 7 and Delphi 2009 installed, then there should be 2 sets of folders in `C:\Program Files\Neos Eureka S.r.l\EurekaLog 7` (substitute with your installation path): `\Lib\Win32\Release\Delphi7` and `\Lib\Win32\Release\Studio12`.

Please refer to IDE names 604 to get "Delphi 2009 <> `Delphi12`" mapping.

If you see EurekaLog's options in IDE 222, but (after enabling EurekaLog for project) compiler says that ExceptionLog7 file is not found (or any other EurekaLog-related file) — then add `C:\Program Files\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Release\Delphi`*`XX`* folder to your library path (Tools\Options\Library), where XX indicate your Delphi version (C ++ Builder and RAD Studio have similar paths).

If you have too many installed components or 3rd party extensions - you `%PATH%` environment variable may become too large. The limit on environment block could prevent some packages from being loaded in IDE. Try to uninstall unused components and libraries.

If you was unable to install EurekaLog automatically - you can install it manually 598.

See also:

- List of all known installation problems in our Knowledge base
- Troubleshooter guide
- Manual installation 598
- Where to find EurekaLog 603
- EurekaLog IDE Setup 604
- IDE names mapping 604
- Uninstallation problems 605
- EurekaLog files layout 619

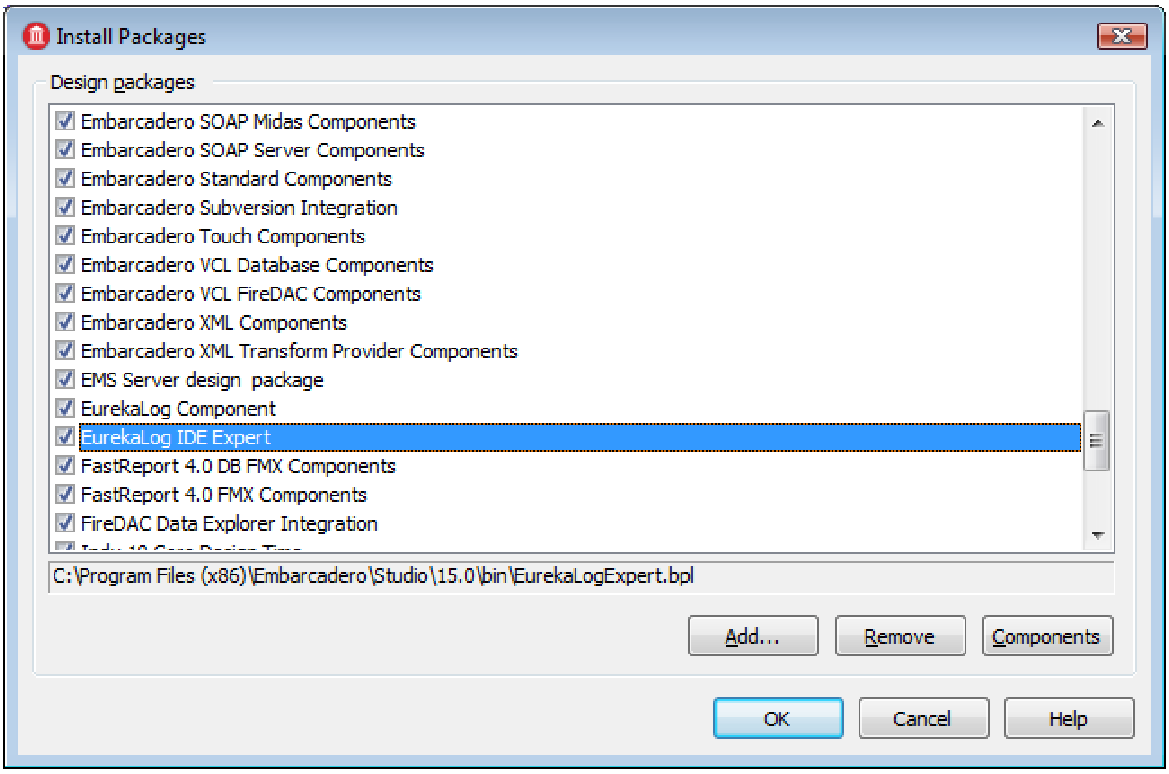
12.2.1 Manual installation

Sometimes there is need to install EurekaLog manually. You may need manual installation to use EurekaLog on build server without running EurekaLog installer. Or you may use manual installation to troubleshoot installation problems. If you're reading this article to solve installation issues - please read this article 596 first.

This article will use the following notation:
`%EUREKALOG%` - installation folder of EurekaLog. For example: `C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\`
`%IDE%` - installation folder of your IDE. For example: `C:\Program Files (x86)\Embarcadero\Studio\15.0\`

IDE Expert

1. Determinate name and integer version of your IDE by using this table 604. For example: `Studio16` for name and `16` for integer version. This article will use `%IDENAME%` to mark name of your IDE and `%VER%` for integer version determinated on this step.
2. Check if you have `%EUREKALOG%\Packages\%IDENAME%\` folder. If yes - skip to step 3. If no - run EurekaLog installer with /Force_D%VER% and/or /Force_C%VER% command-line switches 31.
3. Copy `EurekaLogCore%VER%0.bpl`, `.dbg`, `.jdbg` files from `%EUREKALOG%\Packages\%IDENAME%\` folder to 32-bit `Windows\System32` folder. It is a run-time package (EurekaLogCore). This file will be used by IDE in steps 7 and 9 below.
4. Run your IDE.
5. Go to `Components/Install packages` menu item of your IDE.
6. Find any EurekaLog packages in the list. Remove them (if present).
7. Click on "Add" button and pick up `%EUREKALOG%\Packages\%IDENAME%\EurekaLogExpert%VER%0.bpl` file.
8. Make sure that "EurekaLog IDE Expert" appears in package list and it has checkbox checked.
9. Click on "Add" button and pick up `%EUREKALOG%\Packages\%IDENAME%\EurekaLogComponent%VER%0.bpl` file.
10. Make sure that "EurekaLog Component" appears in package list and it has checkbox checked.
11. Close Packages dialog.



Install Packages
Design packages
☑ Embarcadero SOAP Midas Components
☑ Embarcadero SOAP Server Components
☑ Embarcadero Standard Components
☑ Embarcadero Subversion Integration
☑ Embarcadero Touch Components
☑ Embarcadero VCL Database Components
☑ Embarcadero VCL FireDAC Components
☑ Embarcadero XML Components
☑ Embarcadero XML Transform Provider Components
☑ EMS Server design package
☑ EurekaLog Component
☑ EurekaLog IDE Expert
☑ FastReport 4.0 DB FMX Components
☑ FastReport 4.0 FMX Components
☑ FireDAC Data Explorer Integration
C:\Program Files (x86)\Embarcadero\Studio\15.0\bin\EurekaLogExpert.bpl
Add...
Remove
Components
OK
Cancel
Help

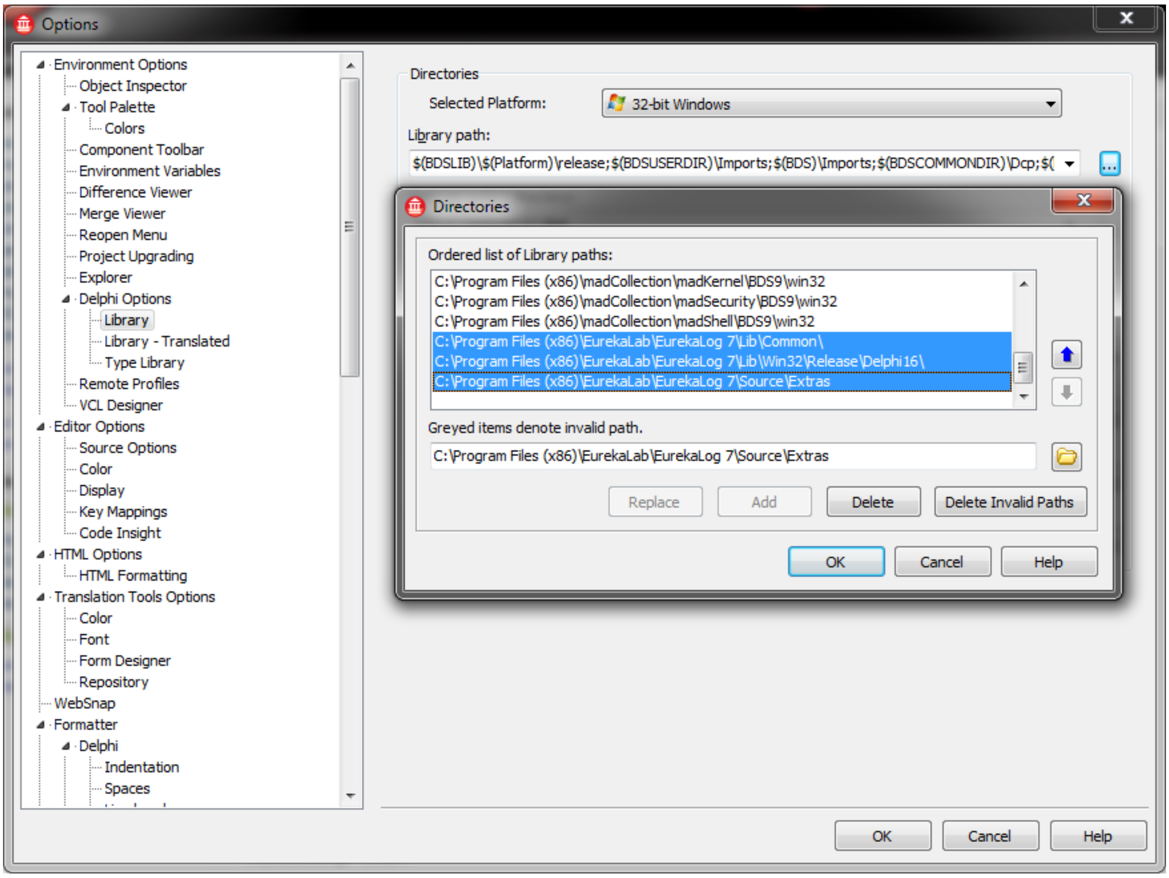
Properly installed IDE expert (Delphi XE7)

Now you should be able to see EurekaLog IDE menu items 222. There also should be TEurekaLogEvents component present on "EurekaLog" tab in component palette.

Note: all installed components and extensions are limited by limitations of 32-bit processes. If your IDE crash after installing and/or using EurekaLog IDE expert - try to remove unused components/extensions.

Library search paths

12. Go to Tools/Options menu item of your IDE (some IDE versions names this menu item as "Environment options").
13. Find Library category in options dialog (it should be a tab for old IDE versions or tree view item for new IDE versions). For C++ Builder as part of RAD Studio - look for C++ Options/Paths category.
14. Find "Library Search Paths" options (other possible name is "Library path").
15. Click on "..." button to open list of library search paths.
16. Remove all EurekaLog-related paths (if any present).
17. Add %EUREKALOG%\Lib\Win32\Release\%IDENAME%\, %EUREKALOG%\Lib\Common and %EUREKALOG%\Source\Extras\ folders.



Selected Platform: 32-bit Windows

Library path: $(BDSLIB)\$(Platform)\release;$(BDSUSERDIR)\Imports;$(BDS)\Imports;$(BDSCOMMONDIR)\Dcp;$(

Ordered list of Library paths:

```
C:\Program Files (x86)\madCollection\madKernel\BDS9\win32
C:\Program Files (x86)\madCollection\madSecurity\BDS9\win32
C:\Program Files (x86)\madCollection\madShell\BDS9\win32
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common\
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Win32\Release\Delphi16\
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Source\Extras
```

Greyed items denote invalid path.

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Source\Extras

Properly registered library paths (Delphi XE2)

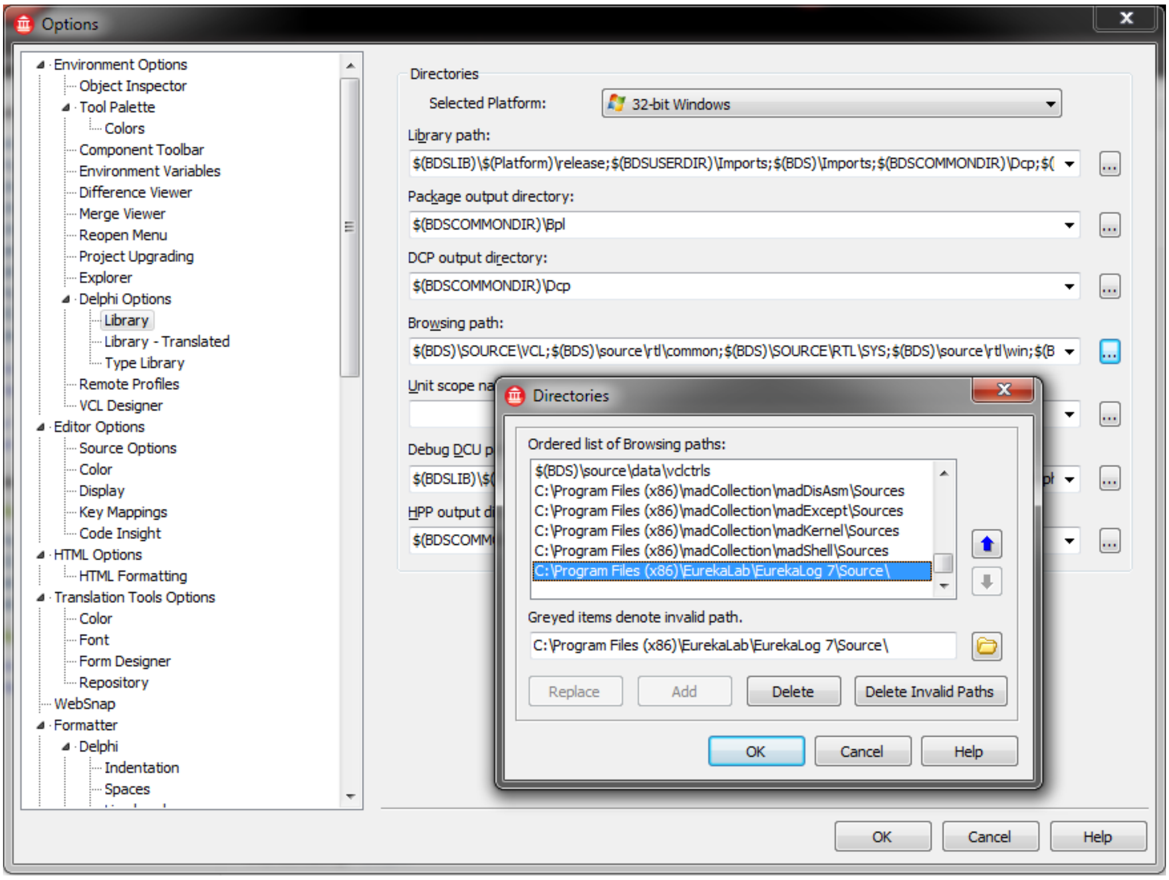
Now you should be able to compile your projects.

18. (Optional, Delphi XE2+ only) Change platform to "64-bit Windows" and repeat steps 12-14, only use Win64 instead of Win32 folder.

Note: search paths may be limited in length. If your changes aren't working - please remove unused paths or move EurekaLog towards the beginning of the list.

Source browsing paths

19. Find "Browsing path" option in the same dialog.
20. Click on "..." button to open list of source browsing paths.
21. Remove all EurekaLog-related paths (if any present).
22. Add `%EUREKALOG%\Source\` folder.
23. (Optional) Add all subfolders of `%EUREKALOG%\Source\` folder.



Directories

Selected Platform: 32-bit Windows

Library path: $(BDSLIB)\$(Platform)\release;$(BDSUSERDIR)\Imports;$(BDS)\Imports;$(BDSCOMMONDIR)\Dcp;$(

Package output directory: $(BDSCOMMONDIR)\Bpl

DCP output directory: $(BDSCOMMONDIR)\Dcp

Browsing path: $(BDS)\SOURCE\VCL;$(BDS)\source\rtl\common;$(BDS)\SOURCE\RTL\SYS;$(BDS)\source\rtl\win;$(B

Unit scope na

Debug DCU p $(BDSLIB)\$(

HPP output di $(BDSCOMM

Ordered list of Browsing paths:

```
$(BDS)\source\data\vdctrls
C:\Program Files (x86)\madCollection\madDisAsm\Sources
C:\Program Files (x86)\madCollection\madExcept\Sources
C:\Program Files (x86)\madCollection\madKernel\Sources
C:\Program Files (x86)\madCollection\madShell\Sources
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Source\
```

Greyed items denote invalid path.

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Source\

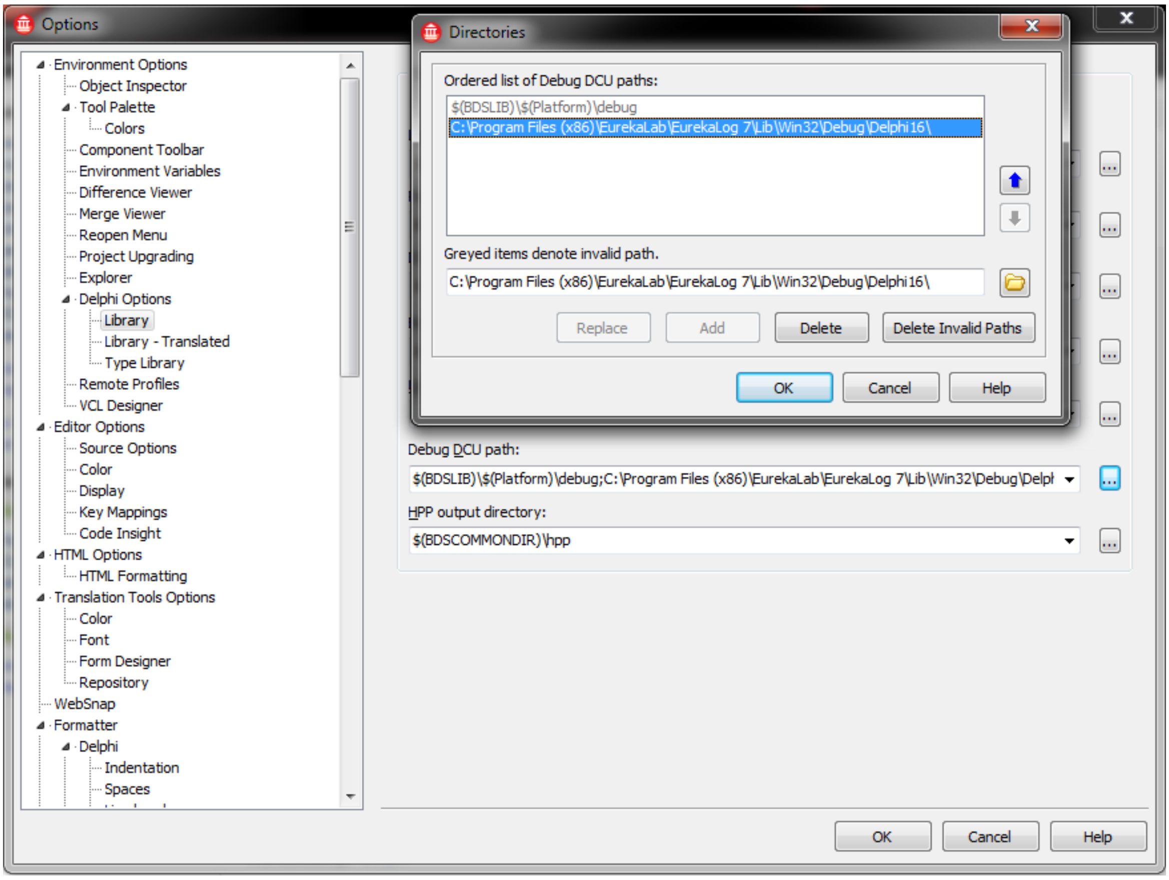
Properly registered (minimal) browsing paths (Delphi XE2)

21. (Optional, Delphi XE2+) Change platform to "64-bit Windows" and repeat steps 17-20.

Note: search paths may be limited in length. If your changes aren't working - please remove unused paths or move EurekaLog towards the beginning of the list.

Debug paths (Delphi only)

24. Find "Debug DCU paths" option.
25. Click on "..." button to open list of debug paths.
26. Remove all EurekaLog-related paths (if any present).
27. Add `%EUREKALOG%\Lib\Win32\Debug\%IDENAME%\` folder.



Ordered list of Debug DCU paths:

$(BDSLIB)\$(Platform)\debug
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Win32\Debug\Delphi16\

Greyed items denote invalid path.

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Win32\Debug\Delphi16\

Debug DCU path:

$(BDSLIB)\$(Platform)\debug;C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Win32\Debug\Delpl

HPP output directory:

$(BDSCOMMONDIR)\hpp

Properly registered debug path (Delphi XE2)

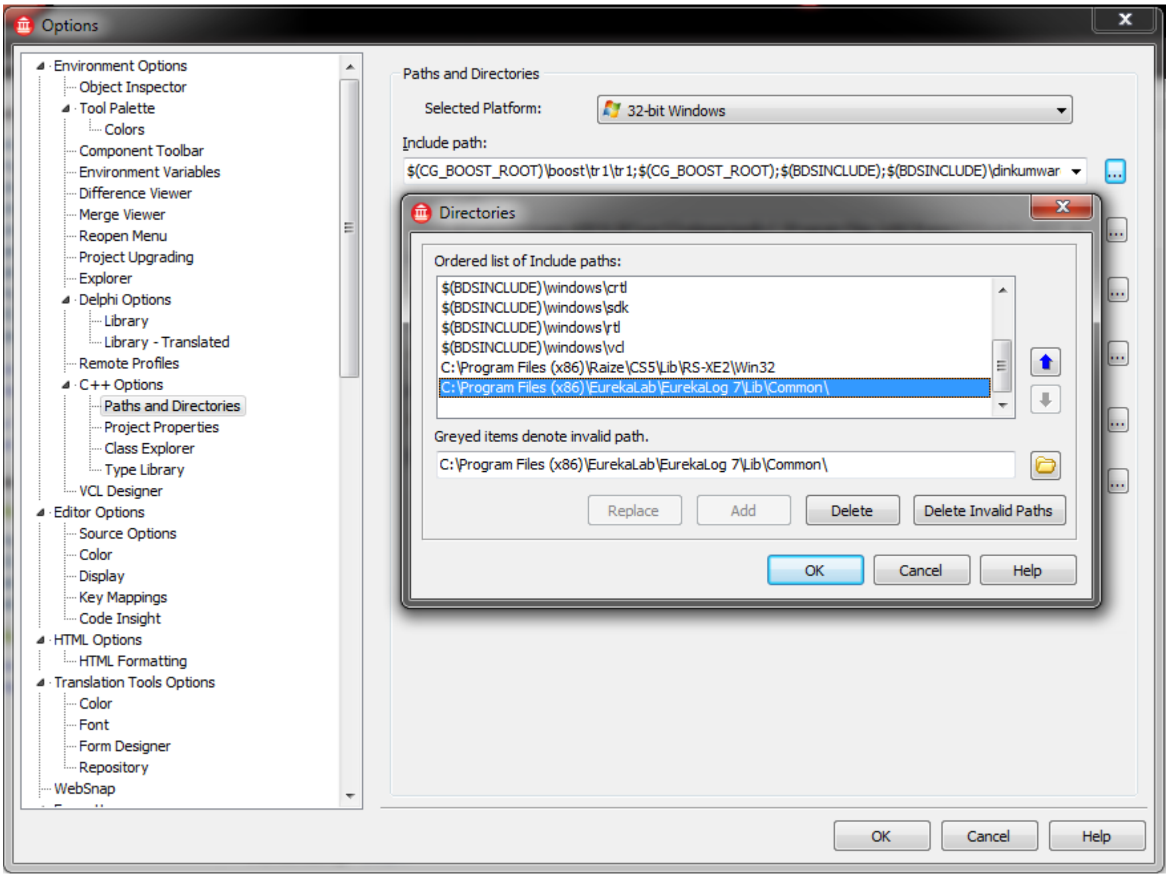
Now you will be able to use 2 sets of files (debug/release). When you enable "Use Debug DCUs" option - files from `Debug` folder will be used. Otherwise ("Use Debug DCUs" option unchecked) - `Release` folder will be used.

28. (Optional, Delphi XE2+) Change platform to "64-bit Windows" and repeat steps 23-25.

Note: search paths may be limited in length. If your changes aren't working - please remove unused paths or move EurekaLog towards the beginning of the list.

Include paths (C++ Builder only)

29. Find "Include paths" option.
30. Click on "..." button to open list of include paths.
31. Remove all EurekaLog-related paths (if any present).
32. Add `%EUREKALOG%\Lib\Common\` folder.



Paths and Directories

Selected Platform: 32-bit Windows

Include path: $(CG_BOOST_ROOT)\boost\tr1\tr1;$(CG_BOOST_ROOT);$(BDSINCLUDE);$(BDSINCLUDE)\dinkumwar

Ordered list of Include paths:

$(BDSINCLUDE)\windows\crtl
$(BDSINCLUDE)\windows\sdk
$(BDSINCLUDE)\windows\rtl
$(BDSINCLUDE)\windows\vcl
C:\Program Files (x86)\Raize\CS5\Lib\RS-XE2\Win32
C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common\

Greyed items denote invalid path.

C:\Program Files (x86)\EurekaLab\EurekaLog 7\Lib\Common\

Properly registered include path (C++ Builder XE2)

33. (Optional, Delphi XE2+) Change platform to "64-bit Windows" and repeat steps 26-28.

Note: search paths may be limited in length. If your changes aren't working - please remove unused paths or move EurekaLog towards the beginning of the list.

Verify project settings

34. Open your project.
35. Make sure IDE expert is not disabled for this project - go to `Components/Install packages` menu item and make sure EurekaLog Expert package is present and enabled.
36. Check project options (`Project/Options`) - make sure there are no EurekaLog paths listed in any options.
37. Delete all .dcu/.obj files of your project. Make a full build.

See also:

- EurekaLog IDE setup 604
- EurekaLog files layout 619
- Where to find EurekaLog 603
- IDE names mapping 604

12.2.2 Where to find EurekaLog

EurekaLog creates files and references at the following locations:

1. **Installation directory**. By default it is `%ProgramFiles%\Neos Eureka S.r.l\EurekaLog 7\` folder (32-bit Program Files folder is used). This folder includes the following subfolders for your IDE:
   - `\Lib\Win32\Debug\`*`Your-IDE`*`\`
   - `\Lib\Win32\Release\`*`Your-IDE`*`\`

   - `\Source\`

2. **Start Menu folder**. By default it is `%Programs%\EurekaLog 7\` folder.
3. **ApplicationData folder**. It is `%AppData%\Neos Eureka S.r.l\EurekaLog\` folder.
4. **Software registry key**. It is `HKLM\Software\EurekaLab\EurekaLog\7.0\` and `HKCU\Software\EurekaLab\EurekaLog\7.0\`.
5. **IDE menu items**. It is Project\EurekaLog Options and Tools\EurekaLog 222.
6. **File associations**.

Refer to IDE names 604 to know more about *Your-IDE* names.

See also:

- Installation problems 596
- IDE menu items 222
- EurekaLog's files layout 619

12.2.3 EurekaLog IDE Setup

EurekaLog performs the following steps to install itself for each IDE:

1. For each IDE EurekaLog installs a **set of dcu/obj/hpp** files into `%EurekaLog%\Lib\Win32\Debug\`*`Your-IDE`*`\` and `%EurekaLog%\Lib\Win32\Release\`*`Your-IDE`*`\` folders.
2. EurekaLog copies **EurekaLogCore*VER*.bpl**, .dbg, .jdbg files from `%EurekaLog%\Packages\`*`Your-IDE`*`\` folder to `Windows\System32` folder (of corresponded bitness).
3. EurekaLog **registers itself into IDE**:
   - Adds EurekaLogExpert 221 *VER* 221 .bpl package 221 (Component\Install package);
   - Adds EurekaLogComponent*VER*.bpl package (Component\Install package);
   - Adds `%EurekaLog%\Lib\Win32\Release\`*`Your-IDE`*`\` to Library paths;
   - Adds `%EurekaLog%\Lib\Win32\Debug\`*`Your-IDE`*`\` to Debug DCU paths;
   - Adds `%EurekaLog%\Source\` to Browsing paths.

Refer to IDE names 604 to know more about *Your-IDE* names.

See also:

- Installation problems 596
- IDE menu items 222
- EurekaLog's files layout 619

12.2.4 IDE names mapping

Real IDE name	EurekaLog name	Integer version
Borland Delphi 4	Delphi4	4
Borland Delphi 5	Delphi5	5
Borland Delphi 6	Delphi6	6
Borland C++ Builder 6	Builder6	6
Borland Delphi 7	Delphi7	7
Borland Delphi 2005	Delphi9	9
Borland Developer Studio 2006	Studio10	10
CodeGear RAD Studio 2007	Studio11	11
Embarcadero RAD Studio 2009	Studio12	12
Embarcadero RAD Studio 2010	Studio14	14
Embarcadero RAD Studio XE	Studio15	15
Embarcadero RAD Studio XE2	Studio16	16
Embarcadero RAD Studio XE3	Studio17	17
Embarcadero RAD Studio XE4	Studio18	18
Embarcadero RAD Studio XE5	Studio19	19
Embarcadero RAD Studio XE6	Studio20	20
Embarcadero RAD Studio XE7	Studio21	21

Real IDE name	EurekaLog name	Integer version
Embarcadero RAD Studio XE8	Studio22	22
Embarcadero RAD Studio 10 Seattle	Studio23	23
Embarcadero RAD Studio 10.1 Berlin	Studio24	24
AppMethod	AppMethod	13

See also:

- Installation problems 596
- EurekaLog IDE setup 604
- Where to find EurekaLog 603

12.3 Uninstallation problems

Note: these are troubleshooting guidelines **for EurekaLog 7 only**. If you uninstall other version of EurekaLog - use corresponding help resources.

A. You should always use EurekaLog's uninstaller, which you could run from Start menu (EurekaLog program group) or from "Uninstall application" Control Panel's applet.

B. If you can't find these items - you can run uninstaller manually by executing its file: `C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\unins000.exe` (exact path depends on your system).

C. However, if uninstaller has failed to remove EurekaLog for some reason - here is a list of steps to do to remove EurekaLog 7 manually:

1. **Unregister EurekaLog**:
   - Close all opened projects in your IDE and select `Component / Install packages` command. Then:
   - o Find "EurekaLog IDE Expert" package and remove it completely (use "Remove" button, do not just uncheck checkbox);
   - o Find "EurekaLog Component" package and remove it completely (use "Remove" button, do not just uncheck checkbox).
   - Go to `Tools / Options` and select "Library" category. Remove any EurekaLog-related paths from "Library path", "Browsing path" and "Debug DCU path".
   - Remove any EurekaLog-related files or folder from any other tools. For example, use `Options / Configure symbols` command in Process Explorer tool, if you've setup it to use EurekaLog's files before.
2. **Close** all opened Delphi/C++Builder/RAD Studio/AppMethod instances and Process Explorer tool (or any other debug tool which may use your symbol's cache 230). Alternatively, it's recommended to **restart your PC**.
3. **Delete EurekaLog files**:
   - Delete **EurekaLog's folder**, which is typically "`C:\Program Files\Neos Eureka S.r.l\EurekaLog 7`".
   - Delete **EurekaLogCore*.*, EurekaLogExpert*.*, EurekaLogComponent*.*, ecc32.* and emake.*** files *from "bin" folder* of your IDE.
   - Delete **EurekaLogCore*.*** files from `Windows\System32` folder (both 32-bit and 64-bit).
   - Delete **EurekaLog's program group** in Start menu. You can do this by opening Start menu and right-clicking on "EurekaLog" program group.
4. **Delete EurekaLog registry settings**:
   - Delete **`HKEY_CURRENT_USER\Software\EurekaLab\EurekaLog\7.0`** registry key.
   - Delete **`HKEY_LOCAL_MACHINE\Software\EurekaLab\EurekaLog\7.0`** registry key.
   - (Optionally) You can delete the entire `\Software\EurekaLab\` key, if you want to.
   - Delete **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{BB108562-6527-4FA0-8928-6F5ECDBB08CF}_is1`** registry key (note: it's `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{BB108562-6527-4FA0-8928-6F5ECDBB08CF}_is1` on 64bit machines).
5. **Delete your data files**:
   - Delete **%APPDATA%\Neos Eureka S.r.l\EurekaLog\** folder. For example, it can be `C:\Users\UserName\AppData\Roaming\Neos Eureka S.r.l\EurekaLog\` folder.

- (Optionally) You can delete the entire %APPDATA%\Neos Eureka S.r.l\ folder, if you want to.
- Delete **each .dcu/*.obj files** of your projects, which could be rebuilded from .pas/.cpp files.

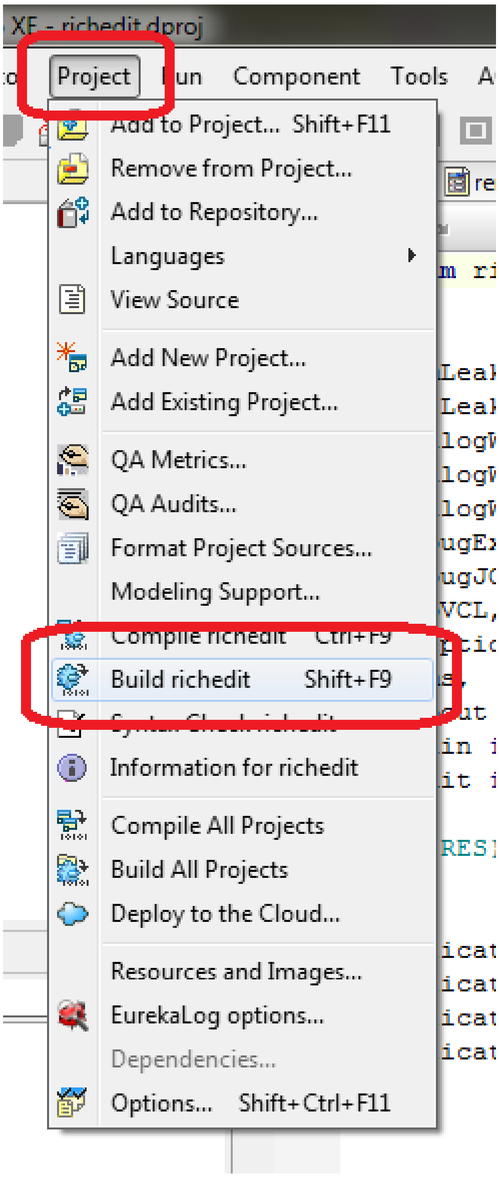
6. **Delete any left-overs**:
   - Run a search on your hard drive and look for any ***.elf**, ***.eof**, ***.etf**, **EurekaLog***, **ecc32.***, **emake.***, **ExceptionLog*** files. If you've found something and you don't need this - delete it.

See also:

- Installation problems 596
- EurekaLog's files and data location 603
- EurekaLog installation into IDE 604

12.4 Enabling EurekaLog problems

When you have enabled EurekaLog for your project 33 - you must build your project:

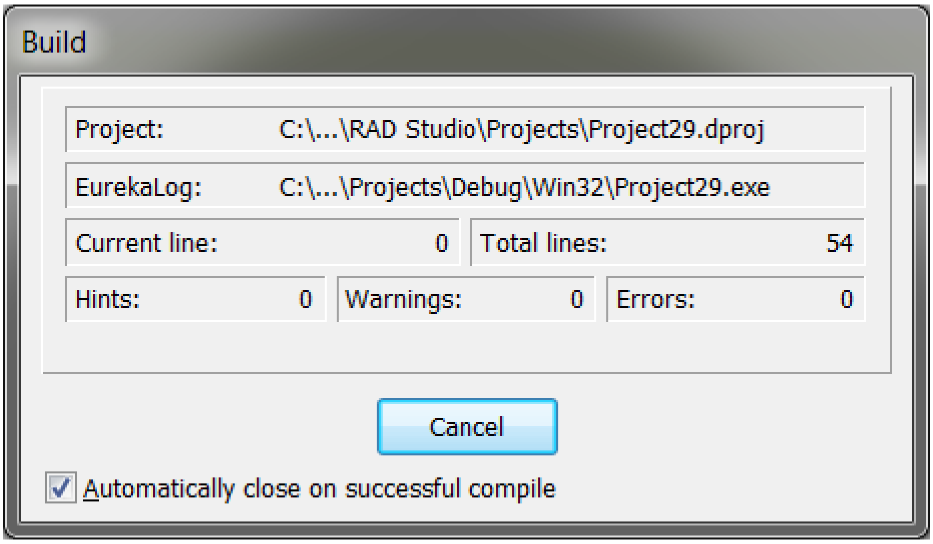


Build your project

You will see a standard processing window, where you can see "Compiling" and "Linking" stages.

See also: Compiling your project with EurekaLog 421.

You should see **one additional stage** at the very end of this process. This stage is called "post-processing 426". "EurekaLog":



Build

Project: C:\...\RAD Studio\Projects\Project29.dproj

EurekaLog: C:\...\Projects\Debug\Win32\Project29.exe

Current line: 0 Total lines: 54

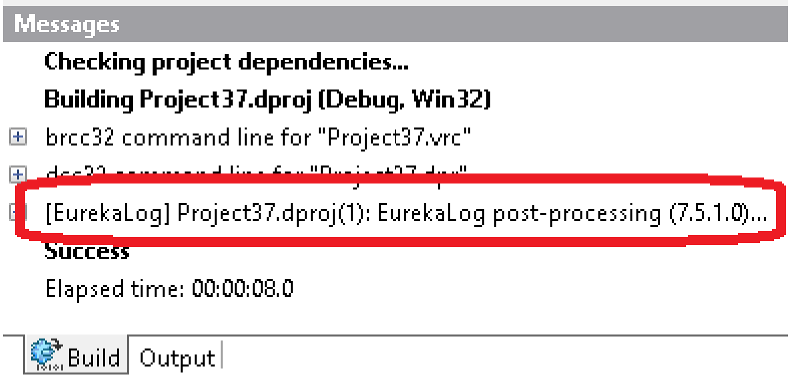
Hints: 0 Warnings: 0 Errors: 0

☑ Automatically close on successful compile

EurekaLog's post-processing during compilation

It indicates that EurekaLog is active, enabled and doing post-processing of your executable.

Additionally, EurekaLog's post-processing stage will be listed in IDE compiler output - use IDE's "View" / "Messages" command to open it:



Messages

Checking project dependencies...

Building Project37.dproj (Debug, Win32)

⊞ brcc32 command line for "Project37.vrc"

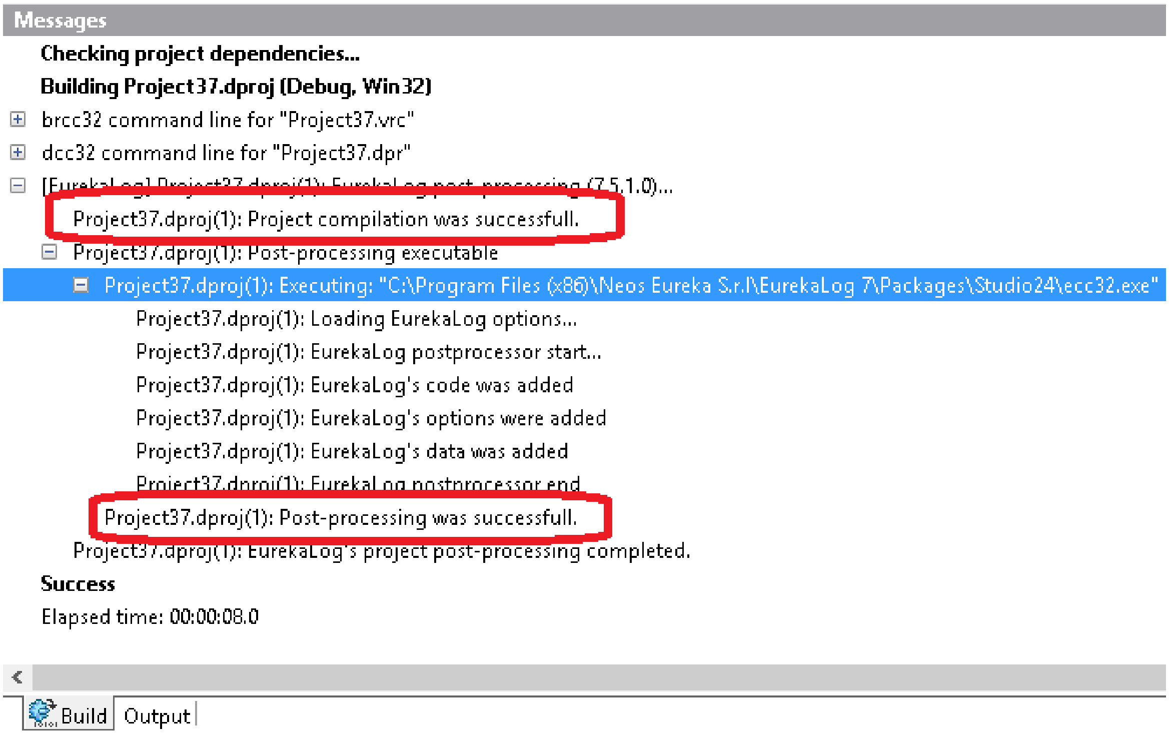
⊞ dcc32 command line for "Project37.dpr"

⊞ [EurekaLog] Project37.dproj(1): EurekaLog post-processing (7.5.1.0)...

Success

Elapsed time: 00:00:08.0

"EurekaLog" / "post-processing" stage in IDE output



```
Checking project dependencies...
Building Project37.dproj (Debug, Win32)
⊞ brcc32 command line for "Project37.vrc"
⊞ dcc32 command line for "Project37.dpr"
⊟ [EurekaLog] Project37.dproj(1): EurekaLog post-processing (7.5.1.0)...
    Project37.dproj(1): Project compilation was successfull.
  ⊟ Project37.dproj(1): Post-processing executable
    ⊟ Project37.dproj(1): Executing: "C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\Packages\Studio24\ecc32.exe"
        Project37.dproj(1): Loading EurekaLog options...
        Project37.dproj(1): EurekaLog postprocessor start...
        Project37.dproj(1): EurekaLog's code was added
        Project37.dproj(1): EurekaLog's options were added
        Project37.dproj(1): EurekaLog's data was added
        Project37.dproj(1): EurekaLog postprocessor end
      Project37.dproj(1): Post-processing was successfull.
    Project37.dproj(1): EurekaLog's project post-processing completed.
Success
Elapsed time: 00:00:08.0
```

Same window expanded - this is output from a normal successful compilation

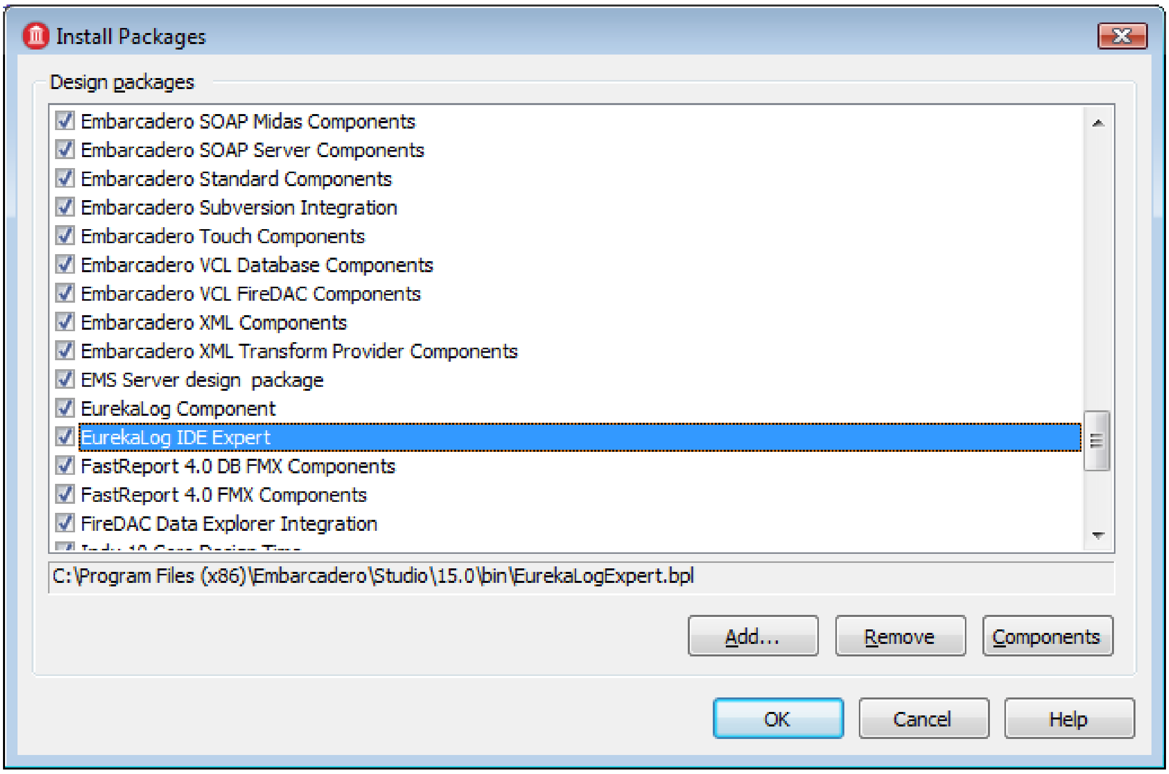
If you don't see this post-processing stage and/or IDE messages output is empty - then something is wrong.

Most common problem

You have dropped TEurekaLogEvents component on the form but did not enabled EurekaLog for your project via "Project" / "EurekaLog Options" menu. TEurekaLogEvents component is supposed to react on EurekaLog's events, but it does nothing if EurekaLog is not added. Please, enable EurekaLog for your project as explained here 33.

Make sure that:

1. **EurekaLog IDE expert is installed**:



Install Packages

Design packages

- ☑ Embarcadero SOAP Midas Components
- ☑ Embarcadero SOAP Server Components
- ☑ Embarcadero Standard Components
- ☑ Embarcadero Subversion Integration
- ☑ Embarcadero Touch Components
- ☑ Embarcadero VCL Database Components
- ☑ Embarcadero VCL FireDAC Components
- ☑ Embarcadero XML Components
- ☑ Embarcadero XML Transform Provider Components
- ☑ EMS Server design package
- ☑ EurekaLog Component
- ☑ EurekaLog IDE Expert
- ☑ FastReport 4.0 DB FMX Components
- ☑ FastReport 4.0 FMX Components
- ☑ FireDAC Data Explorer Integration

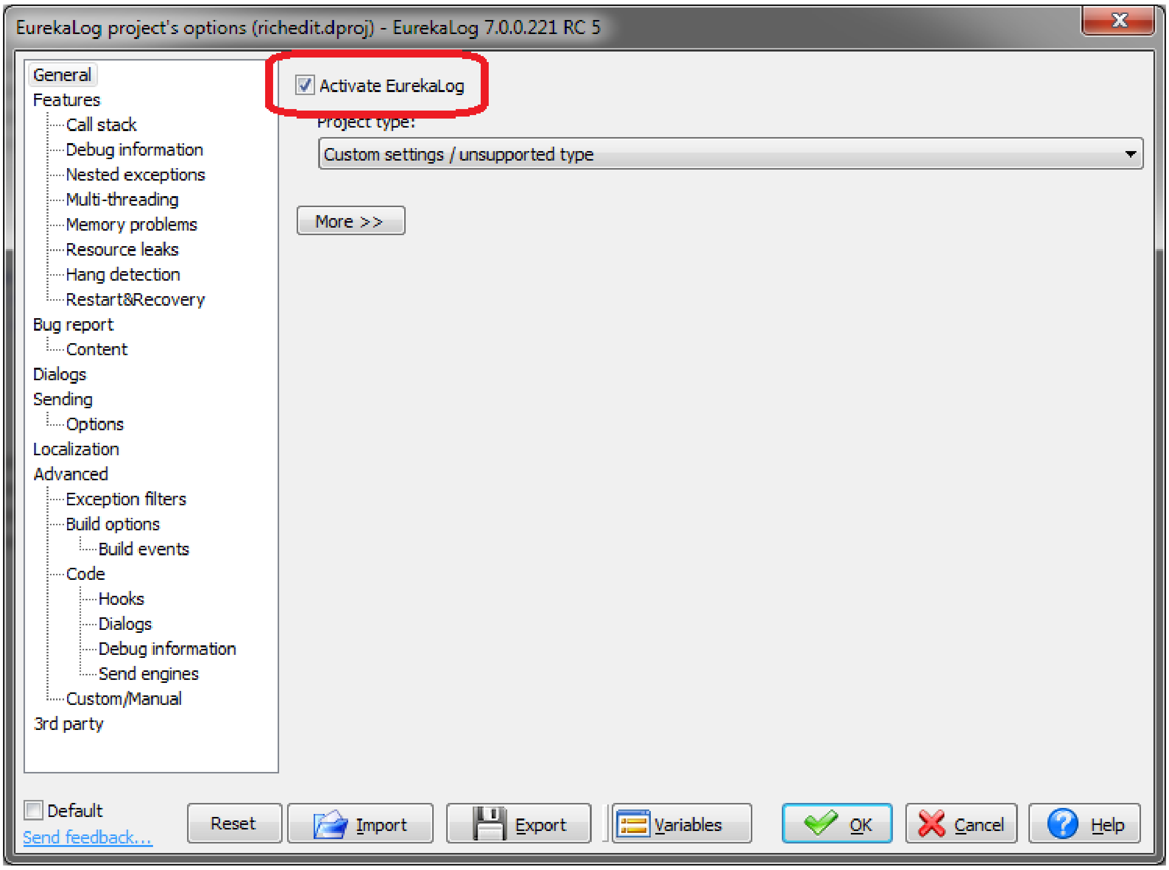
C:\Program Files (x86)\Embarcadero\Studio\15.0\bin\EurekaLogExpert.bpl

Verifying that EurekaLog IDE expert is installed

You should find it into "Components" \ "Install packages" IDE menu. If EurekaLog IDE Expert package is disabled - enable it. If it's not even there - click on "Add" button and pick `EurekaLogExpert.bpl` package from \Packages*IDEName*\ subfolder of your EurekaLog installation. Replace *`IDEName`* with your real IDE version. Use IDE name mapping 604 to determinate which folder corresponds to your IDE.

If you can't find `EurekaLogExpert.bpl` file - then you need to reinstall EurekaLog and make sure that you have selected support for your IDE during installation. Please, follow these guidelines 596.

2. **EurekaLog is enabled** in your project:


EurekaLog project's options (richedit.dproj) - EurekaLog 7.0.0.221 RC 5

☑ Activate EurekaLog

Project type: Custom settings / unsupported type

Enabling EurekaLog in your project

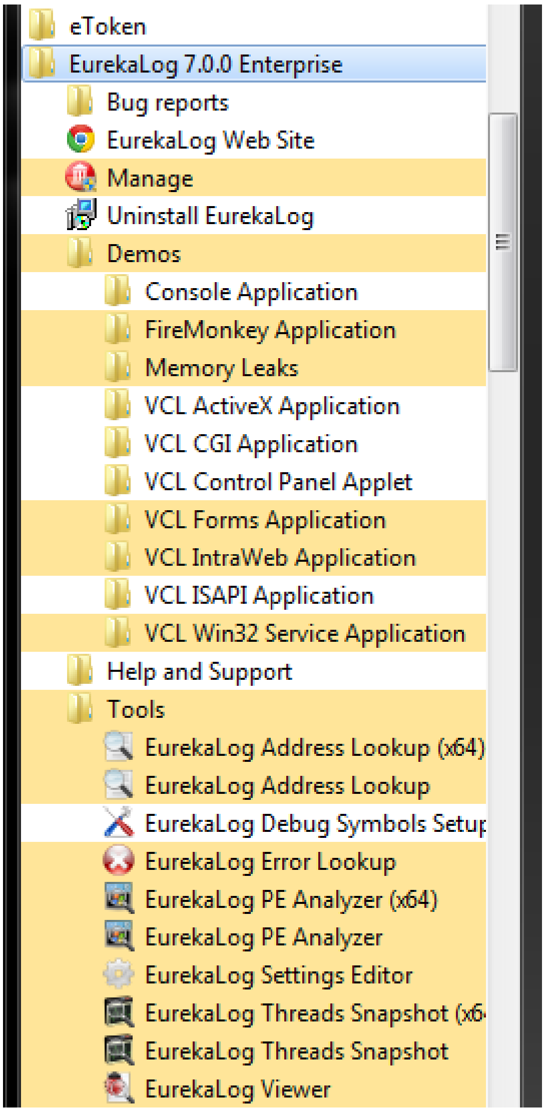
Please, follow these guidelines 33. **Make sure to apply proper application type 363 for your project!**

Once you've checked and verified both items - make final confirm that EurekaLog was really injected into your executable 610.

12.4.1 Verifying that EurekaLog was enabled

If both conditions 606 (EurekaLog is enabled in your application AND EurekaLog IDE Expert is installed) are true - then everything should be OK (see also: compilation from command line 421). You can confirm this by analyzing your executable module after compilation.

Open **Start** / **Programs** and find EurekaLog folder:

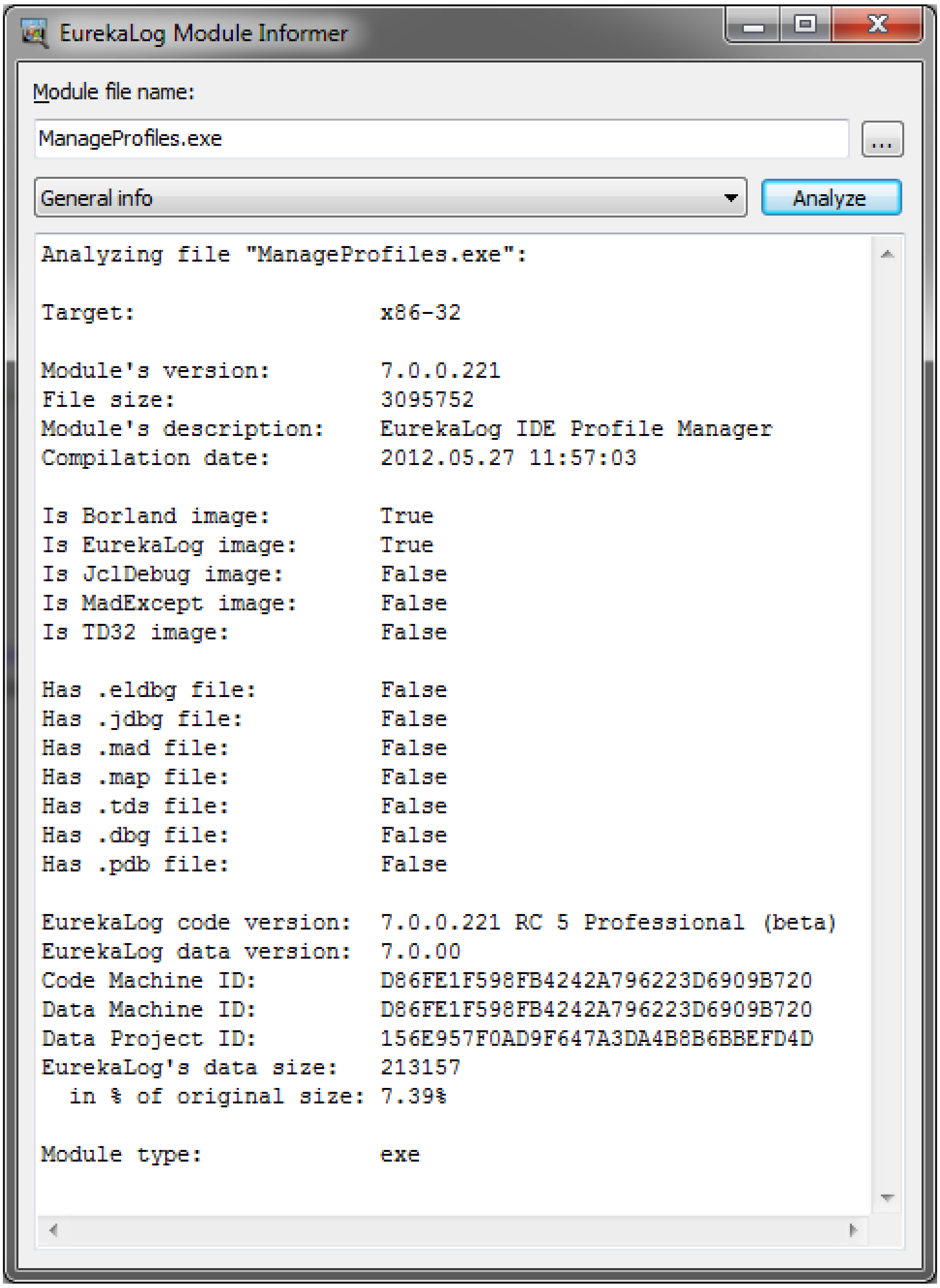


EurekaLog's menu items in Start menu

Find the "**EurekaLog PE Analyzer**" tool and run it (it should be in the "Tools" subfolder). Select your executable file with the help of "..." button and click on "Analyze" button.

Note: you can also run PE Analyzer tool via **Tools \ EurekaLog \ PE Analyzer** menu item in IDE (see also 222).

If your application was compiled properly, then you should see "**Is EurekaLog image: True**" and information about EurekaLog version:



```
Analyzing file "ManageProfiles.exe":

Target:                   x86-32

Module's version:         7.0.0.221
File size:                3095752
Module's description:     EurekaLog IDE Profile Manager
Compilation date:         2012.05.27 11:57:03

Is Borland image:         True
Is EurekaLog image:       True
Is JclDebug image:        False
Is MadExcept image:       False
Is TD32 image:            False

Has .eldbg file:          False
Has .jdbg file:           False
Has .mad file:            False
Has .map file:            False
Has .tds file:            False
Has .dbg file:            False
Has .pdb file:            False

EurekaLog code version:   7.0.0.221 RC 5 Professional (beta)
EurekaLog data version:   7.0.00
Code Machine ID:          D86FE1F598FB4242A796223D6909B720
Data Machine ID:          D86FE1F598FB4242A796223D6909B720
Data Project ID:          156E957F0AD9F647A3DA4B8B6BBEFD4D
EurekaLog's data size:    213157
  in % of original size:  7.39%

Module type:              exe
```

Typical output from Module Informer for correct compiled file

Make sure that this information matches your expectations:
1. **Compilation date** is now (i.e. you're not looking into old file);
2. "**Is EurekaLog image**" = True;
3. **EurekaLog version** matches the installed version of EurekaLog (i.e. your installation is not messed up with new and old files);
4. There are EurekaLog units inside (use detalization level "Units");
5. EurekaLog's **code and data versions** match each other (up to first 3 numbers); EurekaLog code version must be higher or equal to version of data format. E.g. code = "7.0.3.0" and data ="7.0.01" is OK, but code = "7.0.1.0" and data = "7.0.02" is not;
6. EurekaLog's code and data **machine IDs** match each other;

You can also change "General info" to other (more descriptive) positions. Thus, you can inspect injected EurekaLog options, used units and other information.

If there is no problem with compiled executable, but it still doesn't work as expected - then you can enable debug mode to see what's going on 613.

12.5 EurekaLog run-time problems

Once you have verified that EurekaLog's code and data was added to your executable 610 - you can start to diagnose what is wrong at run-time.

Most common problems

1. You have assigned `Application.OnException` event handler or have used `TApplicationEvents` component (`VCL.AppEvnts` unit).

When you assign `Application.OnException` event (this is also what `VCL.AppEvnts` unit does) - EurekaLog assumes that you want to handle exception by yourself. You may remove your `Application.OnException` handler (and remove `VCL.AppEvnts` unit) or call EurekaLog manually from your handler (call `HandleException` routine from EBase unit).

2. You dropped `TEurekaLogEvents` component on the form but did not enabled EurekaLog for your project via "Project" / "EurekaLog Options" menu. `TEurekaLogEvents` component is supposed to react on EurekaLog's events, but it does nothing if EurekaLog is not added. Please, enable EurekaLog for your project as explained here 33.

Additionally, EurekaLog contains self-debug code which helps you to diagnose problems with EurekaLog itself. Usually this debug code is used when you're not able to debug your application directly. For example, when your application stop working after protecting it with EXE protector tool. If you pack or protect executable - it usually will not work under debugger. That's when you can use debug code to find out the reason.

Prepare

Debug code is present only in debug version of EurekaLog .dcu files. You have to enable debug version before using debug features:

- If you're using precompiled files (default): go to Project / Options and turn on "Use Debug DCUs" option. Alternatively, you can add explicit search path to `C:\Program Files\Neos Eureka S.r.l\EurekaLog 7\Lib\Win32\Debug\Studio11\` (change the path to match your installation and IDE version).
- If you're using recompiled files 620 (for Enterprise only): be sure to enable "`DEBUG_EL_CODE`" conditional symbol in project options.

Normal run

First, make a normal run with normally working application. Run your application with `--el_debug` or `--el_debug_standalone` command line option. This option will turn on debugging code in EurekaLog. Debugging code will capture and log information about EurekaLog's work during application life time.

Notes: logging may affect performance of your application. Your application may run slower than usual. This is a normal behavior.

Locating debug output

EurekaLog will create a `el_debug.csl` or *`your-exe-name`*`_el_debug.csl` file in the same folder as application file. This file will contain all debug output after application's exit. If default place (app's folder) is write protected - then file will be placed to Application Data folder.

If you can not find the debug file - use DebugView tool. Download and run this tool, then run your application with `--el_debug` or `--el_debug_standalone` command line option. The DebugView tool will display location for debug output file.

Note:

- DebugView tool could not be used to capture debug output itself;

- The difference between `--el_debug` and `--el_debug_standalone` command line options is that `--el_debug` option will create a single common log file (`el_debug.csl`), and `--el_debug_standalone` option will create one file for each EurekaLog module (.exe or DLL) in your application (*your-exe-name*`_el_debug.csl`).

Problem run

Now, recompile your application with problematic configuration. Run it and make sure it doesn't work as expected. Then run your application again with `--el_debug` or `--el_debug_standalone` command line option. This will produce a new `.csl` file.

Note: each run will overwrite `.csl` file. Be sure to rename/make a copy before running application next time.

Compare results

You can compare two log files which were captured from working and non-working application. Find the differences. Differences will indicate what was going wrong.

EurekaLog produces `.csl` file - which is compatible with Raize Software's CodeSite File Viewer tool. CodeSite File Viewer tool is shipped with recent versions of IDE, you can install it during IDE installation or via GetIt. Alternatively, you may download a free CodeSite tools from Raize web-site.

Note:
- It is a good idea to organize log messages by threads. Additionally, you may organize by categories.
- EurekaLog streams log to CodeSite-compatible format. However, it is not a replacement for CodeSite. It does not support dispatching (e.g. Live Viewer, Controller, destinations, TCP, etc.), EurekaLog does not have many CodeSite features.

See also:
- Internal errors 591
- Recompiling EurekaLog 619

12.6 Breakpoints

Sometimes breakpoints in your project are not working. This issue is not specific to EurekaLog (usually). This article contains several things which you should to try.

- First, be sure to (re-)setup debugging options as explained here 58. Be sure to enable debug information and be sure not to have directives like {$D-} in your code.
- Check file paths:
  - Be sure that you don't have two different units with same name in different folders (i.e. new and old files). You may load wrong unit in editor, but debugger will operate on another unit.
  - Be sure that output folder for units is listed as library search path (per-project setting).
  - Be sure that your file is included in project and it will be compiled with the project:
    - Check search paths of the project;
    - Check used files (with explicit file paths) in .dpr/.dproj files (Project / View source).
- Delete old files (*.dcu, *.obj, *.lib, *.tds, *.rsm, *.bpl, *.dcp, *.exe, *.dll) and **build** your project(s) (not just make/compile).
- **[EurekaLog]** Disable "Delete service files after compilation" option 234.
- It's possible that you've opened file that do not belong to project. File may be already compiled or it may be not part of the project (think about exe/DLL).
- It's possible that your project is DLL and DLL wasn't loaded by host process. Breakpoints will not work until DLL is loaded. Sometimes DLL may be not loaded at all.
- It's possible that code with breakpoint isn't used in your project. For example, a unit can contain 3 functions. Functions #1 and #2 are called from the program, but function #3 is not. Thus, breakpoints in function #3 will not work. Similarly, optimized compiler may remove individual lines of code from the functions. For example, code line calculates value, but this value is not used anywhere. Such code will be omitted from final executable.

- It's possible that your code is placed in run-time package (.bpl). Thus, rebuilding your project will not affect that unit. And that unit (in package) may be compiled without debug information. If this is the case - then you need to either rebuild package with debug information or remove package from being used in your project.
- It's possible that debugger can not associate running program with your project. To avoid this:
  - Do not rename application and DLLs. Name project files with desired names.
  - Do not move application and DLLs to other folders. Setup proper output folder instead of moving files. This is especially true for DLLs and Win32 services.
  - Do not run application (don't load DLLs) via alternative names. For example, the same folder can be accessed via different names (folder itself, hardlink, via mapped network share, via reparse point, subst-drive). So, if you can access the same file/folder with different names - be sure to use exact same name in all places. This is especially true for DLLs, because they are loaded by host process (which may be not controlled by you). You can use Process Monitor or similar tool to check file accesses.
  - Be sure that all file paths contain only ASCII symbols (latin). Do not use local symbols with codes above 128.
  - Do not alter date/time (date of creation or modification) of compiled files and .dcu/.obj/.lib files. Changing date/time may cause debugger to consider files being changed. "Changed" files requires recompile, thus they do not match each other. Sometimes date/time can be changed by anti-virus software or post-processing tools (like digital signing).
  - Do not delete and do not move .dcu/.obj/.lib files. Be sure to setup unit output folder to the right place. Be sure that unit output folder is listed in library search paths. Try to place .dcu/.obj/.lib files in same folder as .exe/.dll/.bpl/.dcp files.
  - Be sure that all files can be found. Specify search paths or include unit explicitly via "Project / Add to project" command.
  - Try to enable TD32 debug information.
  - As extreme case - try to output all files in single folder. I.e. output .dcu/.obj/.lib/.exe/.dll/.bpl/.dcp files in folder with source files.

12.8 Other problems

Please contact our support 16, if you wasn't able to solve your issues.

Part XIII

13 Tools

EurekaLog comes with some additional tools:

- **Address Lookup** - can get human-readable textual description of raw address location in executable file. Can be used to browse through various sources of debug information.
- **Debug symbols paths** - allows you to setup store for Microsoft's PDB files. PDB files can be used by EurekaLog or Process Explorer tool to display more information in call stacks for system DLLs.
- **External settings editor** - it's standalone editor for EurekaLog's project settings. Can load/save settings in .eof format and any supported IDE project type.
- **Module informer** (PE Informer) - extracts information about compiled executable file. Useful for diagnostic.
- **Threads snapshot** - dumps information about threads in selected process to file. This includes call stacks and wait chains. Useful for debugging hangs on remote machines without debugger.
- **Viewer** - allows you to view bug reports in more convenient way. It also can work as general database for bug reports (i.e. bug tracker).

Part XIV

14 Recompilation (Enterprise)

This article will explain files layout for EurekaLog 7 installation and recompilation of EurekaLog (applicable only for Enterprise edition).

1. Files layout 619
2. Recompilation 620

See also:
- Customizing EurekaLog 180

14.1 Files layout

EurekaLog 7 changes way to manage its files. Now files are organized in typical manner for Delphi libraries:

- `\Bin` - compiled executables (tools), 32-bit. Each EurekaLog tool and helper has its .exe/.dll files here (except for EurekaLog itself).
- `\Bin64` - compiled executables, 64-bit. Not all tools have 64-bit counterparts.
- `\Lib` - precompiled EurekaLog files (units):
  - `\Lib\<platform>` - precompiled EurekaLog units (.dcu, .obj, .hpp). Each folder has subfolders for Debug/Release configuration and subfolder for different IDE versions.
  - `\Lib\Obj\<platform>` - precompiled external source code (non-Delphi). This is location for .obj files of ZLib project. ZLIb is used by EurekaLog to compress injected debug information.
  - `\Lib\Common` - copy of all resource files.

  `\Lib` folder is used to compile your projects.
- `\Packages` - EurekaLog executables (packages and command-line compiler).
- `\Source` - source code of EurekaLog (.pas files). Full source code is available only for Enterprise edition. Other editions have only headers (declarations). Full source code can be used to recompile EurekaLog. By default this folder is not used by your projects.
- `\Source` - EurekaLog files. Those are files for your application.
- `\Source\Extras` - additional EurekaLog files. Those are add-on files, which are always distributed with source code and they are always recompiled for your applications.
- `\Source\Design` - files for IDE expert. Used at design-time.
- `\Source\Compiler` - files for command-line compiler (ecc32/emake).
- `\Source\Common` - common files (includes and resources).

`\Lib` folder should be mentioned in search paths either in IDE or your project. When you compile your project - precompiled files from `\Lib` folder are used. EurekaLog itself is not compiled (which is faster, saves original settings for EurekaLog compilation, does not modify files, UAC-friendly).

There are `Debug` and `Release` subfolders in `\Lib\<platform>` folders.
`Release` contains files which were compiled with "Use Debug DCUs" option unchecked.
`Debug` contains files which were compiled with "Use Debug DCUs" option checked.
There are no significant differences in other compilation options for these files. Both have debug information included.
`Release` folder (proper IDE subfolder - see below) should be listed in library search paths for IDE or project.
`Debug` folder (proper IDE subfolder - see below) should be listed in debug search paths for IDE or project.

If your application has "Use Debug DCUs" option checked - then files from `Debug` folder will be used (regardless of Debug/Release profile in your application).
If your application has "Use Debug DCUs" option unchecked - then files from `Release` folder will be used (regardless of Debug/Release profile in your application).

Important note: there are known binary incompatibilities between Release and Debug versions of stock RTL. If you compile your unit against Debug RTL's DCUs - it may not work with Release RTL's DCUs. That is why we ship both Release and Debug sets. Release set of our DCUs is compiled against Release RTL's DCUs (and, additionally, have no

DEBUG_EL_CODE defined), Debug set of our DCUs is compiled against Debug RTL's DCUs (and, additionally, have DEBUG_EL_CODE defined). Therefore, if you compile your project in Release profile, but your search paths pick up Debug folder from EurekaLog - you may get incompatibility errors. Again, be sure that library path points to Release profile of EurekaLog, and you have Debug paths points to Debug profile.

Both Release and Debug folders contain subfolders for various supported IDEs. To map your IDE version to name of folder inside Release and Debug folders - see this article 604.

Source folder should be added to IDE browsing paths options, so IDE will be able to open and show EurekaLog's .pas files.

Run-time package (EurekaLogCore) from Packages folder is copied to Windows\System32 folder. This is done to avoid altering %PATH% environment variable.

The entire EurekaLog folder in Program Files contains only read-only files which are not modified for typical usage cases. All files which should be writable (such as Demos, bug reports, profiles, translations, etc.) are installed into %APPDATA%\Neos Eureka S.r.l \EurekaLog\ folder.

See also:
- EurekaLog IDE setup 604
- Where to find EurekaLog 603

14.2 Recompilation

Please read these articles first:
- EurekaLog files layout 619
- EurekaLog IDE setup 604

EurekaLog Enterprise comes with full source code, which you can use to recompile EurekaLog. EurekaLog Enterprise contains additional \Projects folder, which contains project files to compile EurekaLog. There are project files for run-time package, design-time packages (IDE expert, component registration) and command-line compiler (ecc32/emake). There are no special projects for EurekaLog itself, because EurekaLog is not application, but library.

To recompile EurekaLog, you first must delete existing .dcu/.obj files. We recommend to rename all \Lib\<*platform*> folders (please, keep \Lib\Obj\ and \Lib\Common untouched) to remove them from search path, so IDE will not be able to find and use them. That way you will be able to restore original files at any time.

The next step depends on what you want to recompile.

Packages and compilers

1. To recompile packages and/or compilers you must either run your IDE under administrator account (this should be done only once - only for recompilation) or to change file permissions for EurekaLog folder in Program Files to allow modifications by limited user accounts.
2. Then you should open corresponding project file from Projects folder. Please, select project corresponding to your IDE. You can verify it here 604. You can also open project group to load all 3 projects (run-time package, design-time package and command-line compiler).
3. (Optionally) Change settings of project(s) as you desire.
4. Compile projects. Compiled executables will be placed to Packages folder. Compiled units will be placed to corresponding folder under Lib folder (make sure output paths exist).
5. Copy run-time package (EurekaLogCore) files into Windows\System32 folder of corresponding bitness.

Note: alternatively, you may change output path setting of run-time package (EurekaLogCore) project to Windows\System32 folder.

Important note: be sure to compile all packages with exact same settings. I.e. do not compile Core package with Release profile, but Expert package - with Debug profile. Usually, Debug profile contains `DEBUG_EL_CODE` conditional symbols, which enables various "`{$IFDEF DEBUG_EL_CODE} if EL_Debug then DebugDump('ApplicationShowExceptionHandler'); {$ENDIF}`" constructs, which allows you to diagnose EurekaLog without source code. See more about this feature here 613. So, if you compile packages with different settings - it may cause "unit is different version" errors, as well as "entry point not found" errors.

EurekaLog files

A) Total recompilation

EurekaLog files which are used for your applications are located in `\Source` and `\Source\Common` folders. The simplest way to re-create all .dcu/.obj files for these files - is to compile run-time package (EurekaLogCore) as explained above. This will create set of precompiled files in corresponding subfolder under \Lib folder. In some cases you can not use package project for recompilation. Then you just need to create empty project, set it properties as you like and add all .pas files from `\Source` folder. Recompiling this project will rebuild all units.

B) Recompilation on demand

If you do not want one-time recompilation, but want to constantly use actual `.pas` files (so you recompile EurekaLog each time you compile your project(s)) - then you can add a path to `Source` folder to Library/Search paths. You can do this either automatically or manually.

Automatic way:
1. Run Start / Programs / EurekaLog 7 / Manage tool.
2. Find your IDE in the list and expand it.
3. Click on "(Ent. only) Install EurekaLog 7 for development".
4. Restart your IDE.

Manual way:
1. (Optional) Remove existing .dcu/.obj files for EurekaLog (as explained above) and for your project(s).
2. Add `\Source`, `\Source\Common` and `\Lib\Obj\Win32` (or `Win64`) folders to library search paths of your project (or IDE - to make this change global for all projects). You should add these folders to library search paths, not to browsing search path (as it's by default).
3. Compile your project. This will use .pas files from `Source` folder. EurekaLog will be recompiled each time you compiled your project. .dcu/.obj files will be placed to units output folder of your project (as set in its options).

This method can be used to constantly recompile EurekaLog when you changes its source code a lot (i.e. you develop your project and make changes in EurekaLog at the same time).

Note: we highly recommend to set specific unit output folder for your project (see Project/Options/Directories/Unit output folder). The exact location doesn't matter. But it should be writable. That way you will be able to compile EurekaLog's .pas files even under limited user account. Alternative is to change file permissions for EurekaLog installation folder to allow modifications for limited user accounts.

Final notes

EurekaLog does not contain source code for ZLib library. To recompile `.obj` files from `\Lib\Obj` folder - you must obtain source code for ZLib library. It's open-source cross-platform freeware library written in C. See official web-site.

EurekaLog contains only Delphi source code (.pas). C++ Builder support is implemented as compilation of Delphi files with C++ Builder. There are no `.cpp`/`.hpp` files. All shipped `.obj`/`.hpp`/`.lib`/`.bpi` files are auto-generated from `.pas` files. All EurekaLog projects have "Generate C++ files" option turned on.

Emake compiler is the same as ecc32. You can compile ecc32 and rename it to emake.

EurekaLog uses the same ecc32 file for all IDE versions and personalities.

EurekaLog Viewer Tool uses DevExpress components. DevExpress suite is not included with EurekaLog. To recompile Viewer you must have DevExpress components installed. Viewer uses Express QuantumGrid, Express Printing System and ExpressBars.

Part XV

15 Compatibility

Migration guides:

- [4.5: Upgrading from EurekaLog 3.x to EurekaLog 4.x] 637
- [4.5.x: Upgrading from EurekaLog 4.x to EurekaLog 4.5.x] 637
- [5.x: Upgrading from EurekaLog 4.5.x to EurekaLog 5.x] 636
- [6.x: Upgrading from EurekaLog 5.x to EurekaLog 6.x] 633
- [7.x: Upgrading from EurekaLog 6.x to EurekaLog 7.x] 624

15.1 6.x -> 7.x

Please see our upgrade guide 627.

These are the changes from the old 6.x version to the new 7.x version:

1. Old **ExceptionLog** unit is renamed to new **ExceptionLog7** unit. You should replace references to ExceptionLog unit with references to ExceptionLog7 unit.
2. **TEurekaLog** component was renamed to **TEurekaLogEvents** component. It is located in **EComponent** unit. You should delete all TEurekaLog components and create new TEurekaLogEvents component.
3. Code from **ExceptionLog** unit was moved to other units (see below):
    1. **ECallStack**
        - GetCurrentCallStack
        - GetCallStackByLevels
        - GetCallStackByAddresses
    2. **EClasses**
        - TEurekaModuleOptions
    3. **EDebugInfo**
        - GetSourceInfoByAddr
    4. **EModules**
        - GetCompilationDate
        - IsEurekaLogModule
        - GetEurekaLogModuleVersion
        - ModuleInfoByHandle
        - ModuleInfoByAddr
    5. **ESendMail**
        - EurekaLogSendEmail
    6. **ETypes**
        - TEmailSendMode
        - TWebSendMode
        - TExceptionDialogType
        - TEurekaActionType
        - esmNoSend
        - esmShellClient
        - esmEmailClient
        - esmSMTPServer
        - esmSMTPClient
        - wsmNoSend
        - wsmHTTP
        - wsmFTP
        - wsmBugZilla
        - wsmFogBugz
        - wsmMantis
        - edtNone
        - edtMessageBox
        - edtMSClassic

   - edtEurekaLog

4. Some code was removed or changed considerably. You should use new alternatives instead:
   1. TEurekaExceptionRecord -> TEurekaExceptionInfo
   2. TEurekaExceptionRecord.CurrentModuleOptions -> TEurekaExceptionInfo.Options
   3. TEurekaLogErrorCode -> TResponse
   4. TExceptionNotifyProc -> TELEvExceptionNotifyProc
   5. TExceptionActionProc -> TELEvExceptionActionProc
   6. TExceptionErrorProc -> TELEvExceptionErrorProc
   7. TPasswordRequestProc -> TELEvPasswordRequestProc
   8. TCustomDataRequestProc -> TELEvCustomDataRequestProc
   9. TAttachedFilesRequestProc -> TELEvAttachedFilesRequestProc
   10. TCustomWebFieldsRequestProc -> TELEvCustomWebFieldsRequestProc
   11. TCustomButtonClickProc -> TELEvCustomButtonClickProc
   12. TExceptionNotify -> TELEvExceptionNotifyMeth
   13. TExceptionActionNotify -> TELEvExceptionActionMeth
   14. TExceptionErrorNotify -> TELEvExceptionErrorMeth
   15. TPasswordRequestNotify -> TELEvPasswordRequestMeth
   16. TCustomDataRequestNotify -> TELEvCustomDataRequestMeth
   17. TAttachedFilesRequestNotify -> TELEvAttachedFilesRequestMeth
   18. TCustomWebFieldsRequestNotify -> TELEvCustomWebFieldsRequestMeth
   19. TCustomButtonClickNotify -> TELEvCustomButtonClickMeth
   20. ExceptionNotify-> RegisterEventExceptionNotify
   21. HandledExceptionNotify -> RegisterEventExceptionNotify
   22. ExceptionActionNotify -> RegisterEventExceptionAction
   23. ExceptionErrorNotify -> RegisterEventExceptionError
   24. PasswordRequest -> RegisterEventPasswordRequest
   25. CustomDataRequest -> RegisterEventCustomDataRequest
   26. AttachedFilesRequest -> RegisterEventAttachedFilesRequest
   27. CustomWebFieldsRequest -> RegisterEventCustomWebFieldsRequest
   28. CustomButtonClickNotify -> RegisterEventCustomButtonClick
   29. StandardEurekaNotify -> ExceptionManager.StandardEurekaNotify
   30. ShowLastExceptionData -> ExceptionManager.ShowLastExceptionData
   31. GetLastExceptionAddress -> ExceptionManager.LastThreadException.Address
   32. GetLastExceptionObject -> ExceptionManager.LastThreadException.ExceptionObject
   33. StandardEurekaError -> ExceptionManager.StandardEurekaError
   34. ForceApplicationTermination -> use dialog properties in CurrentEurekaLogOptions or .Options property of dialog class
   35. GetLastEurekaLogErrorCode -> TResponse
   36. GetLastEurekaLogErrorMsg -> TResponse
   37. GetLastExceptionCallStack -> ExceptionManager.LastThreadException.CallStack
   38. CallStackToStrings -> use Strings.Assign(CallStack)
   39. SetCustomErrorMessage -> use properties of ExceptionManager.LastThreadException
   40. SaveScreenshot -> Screenshot (EScreenshot unit)
   41. TEurekaModuleOptions.GetCustomizedTexts/SetCustomizedTexts are replaced with .CustomizedTexts property.
   42. TEurekaModuleOptions.loSaveModulesAndProcessesSections -> TEurekaModuleOptions.loSaveModulesSection and .loSaveProcessesSection
   43. TEurekaModuleOptions.sndSendScreenshot -> TEurekaModuleOptions.sndScreenshot
   44. TEurekaModuleOptions.sndShowSuccessFailureMsg -> TEurekaModuleOptions.sndShowSuccessMsg and .sndShowFailureMsg

45. TEurekaModuleOptions.sndUseOnlyActiveWindow -> TEurekaModuleOptions.sndScreenshot
46. TEurekaModuleOptions.AppendLogs -> TEurekaModuleOptions.Send*XYZ*AppendLogs
47. TEurekaModuleOptions.EMailAddresses -> TEurekaModuleOptions.Send*XYZ*Target
48. TEurekaModuleOptions.EMailMessage -> TEurekaModuleOptions.Send*XYZ*Message
49. TEurekaModuleOptions.EMailSendMode -> TEurekaModuleOptions.SenderClasses
50. TEurekaModuleOptions.EMailSubject -> TEurekaModuleOptions.Send*XYZ*Subject
51. TEurekaModuleOptions.ProxyPassword -> TEurekaModuleOptions.Send*XYZ*ProxyPassword
52. TEurekaModuleOptions.ProxyPort -> TEurekaModuleOptions.Send*XYZ*ProxyPort
53. TEurekaModuleOptions.ProxyURL -> TEurekaModuleOptions.Send*XYZ*AProxyURL
54. TEurekaModuleOptions.ProxyUserID -> TEurekaModuleOptions.Send*XYZ*ProxyUserID
55. TEurekaModuleOptions.SMTPFrom -> TEurekaModuleOptions.Send*XYZ*From
56. TEurekaModuleOptions.SMTPHost -> TEurekaModuleOptions.Send*XYZ*Host
57. TEurekaModuleOptions.SMTPPassword -> TEurekaModuleOptions.Send*XYZ*Password
58. TEurekaModuleOptions.SMTPPort -> TEurekaModuleOptions.Send*XYZ*Port
59. TEurekaModuleOptions.SMTPServerPort -> TEurekaModuleOptions.Send*XYZ*Port
60. TEurekaModuleOptions.SMTPUserID -> TEurekaModuleOptions.Send*XYZ*Login
61. TEurekaModuleOptions.TrakerAssignTo -> TEurekaModuleOptions.Send*XYZ*AssignTo
62. TEurekaModuleOptions.TrakerCategory -> TEurekaModuleOptions.Send*XYZ*Category
63. TEurekaModuleOptions.TrakerPassword -> TEurekaModuleOptions.Send*XYZ*Password
64. TEurekaModuleOptions.TrakerProject -> TEurekaModuleOptions.Send*XYZ*Project
65. TEurekaModuleOptions.TrakerTrialID -> TEurekaModuleOptions.Send*XYZ*TrialID
66. TEurekaModuleOptions.TrakerUserID -> TEurekaModuleOptions.Send*XYZ*Login
67. TEurekaModuleOptions.WebPassword -> TEurekaModuleOptions.Send*XYZ*AuthPassword
68. TEurekaModuleOptions.WebPort -> TEurekaModuleOptions.Send*XYZ*Port
69. TEurekaModuleOptions.WebSendMode -> TEurekaModuleOptions.SenderClasses
70. TEurekaModuleOptions.WebSSL -> TEurekaModuleOptions.Send*XYZ*SSL or Send*XYZ*TLS
71. TEurekaModuleOptions.WebURL -> TEurekaModuleOptions.Send*XYZ*URL
72. TEurekaModuleOptions.WebUserID -> TEurekaModuleOptions.Send*XYZ*AuthLogin

5. Set options were removed. Instead each set item is represented as Boolean property:
   1. TEurekaModuleOptions.LogOptions -> TEurekaModuleOptions.loDeleteLogAtVersionChange, TEurekaModuleOptions.loAddComputerNameInLogFileName, etc.
   2. TEurekaModuleOptions.CommonSendOptions -> TEurekaModuleOptions.sndShowSuccessFailureMsg, TEurekaModuleOptions.sndSendEntireLog, etc.
   3. TEurekaModuleOptions.ExceptionDialogOptions -> TEurekaModuleOptions.edoShowCopyToClipOption,

TEurekaModuleOptions.edoUseEurekaLogLookAndFeel, etc.
4. TEurekaModuleOptions.BehaviourOptions -> TEurekaModuleOptions.boPauseBorlandThreads, TEurekaModuleOptions.boHandleSafeCallExceptions, etc.
5. TEurekaModuleOptions.LeaksOptions -> TEurekaModuleOptions.loCatchLeaks, TEurekaModuleOptions.loGroupsSonLeaks, etc.
6. TEurekaModuleOptions.ShowOptions -> TEurekaModuleOptions.soAppStartDate, TEurekaModuleOptions.soUserEmail, etc.
7. TEurekaModuleOptions.CallStackOptions -> TEurekaModuleOptions.csoShowDLLs, TEurekaModuleOptions.csoShowBorlandThreads, etc.
8. TEurekaModuleOptions.CompiledFileOptions -> TEurekaModuleOptions.cfoReduceFileSize, TEurekaModuleOptions.cfoCheckFileCorruption, etc.

Replace *XYZ* in properties names above with name of actual send method. For example, to specify URL for Mantis bug tracker - use TEurekaModuleOptions.SendMantisURL property. The following send method prefixes are available:
1. Bugzilla;
2. Fogbugz;
3. FTP;
4. HTTP;
5. Mantis;
6. MAPI;
7. Shell;
8. SMAPI;
9. SMTPClient;
10. SMTPServer;

EMailSendMode and WebSendMode were replaced with single SenderClasses property, which is supposed to list one, two or more send methods in specific order. Use Finalize(Options.SenderClasses) to remove all assigned send methods. Use Options.AddSenderClass and Options.RemoveSenderClass to add/remove method. You can use old esm*ABC*/wsm*ABC* constants (for example, esmSMTP or wsmMantis).

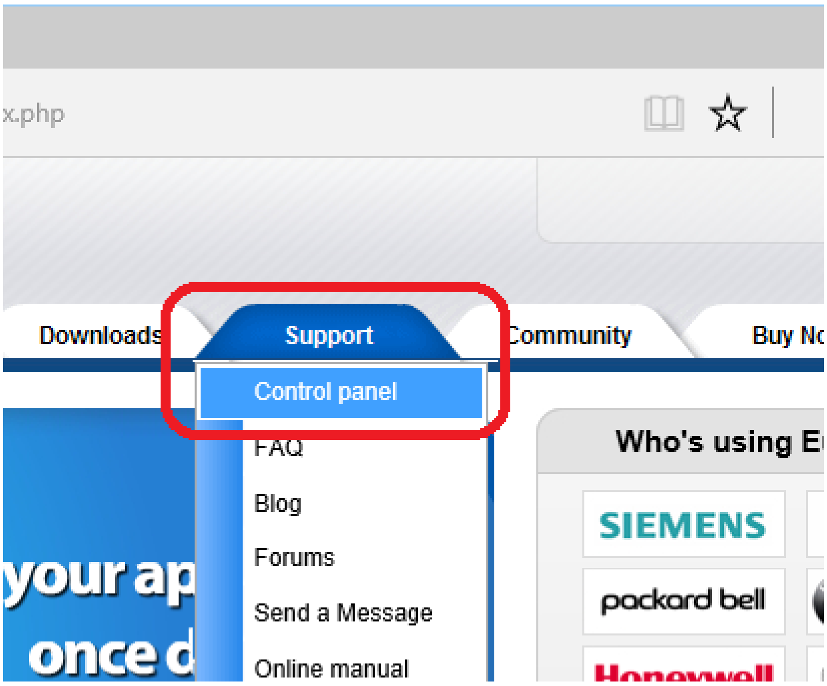
See also:
- Migration guide 627
- What's new in EurekaLog 6.0 634
- What's new in EurekaLog 6.1 634

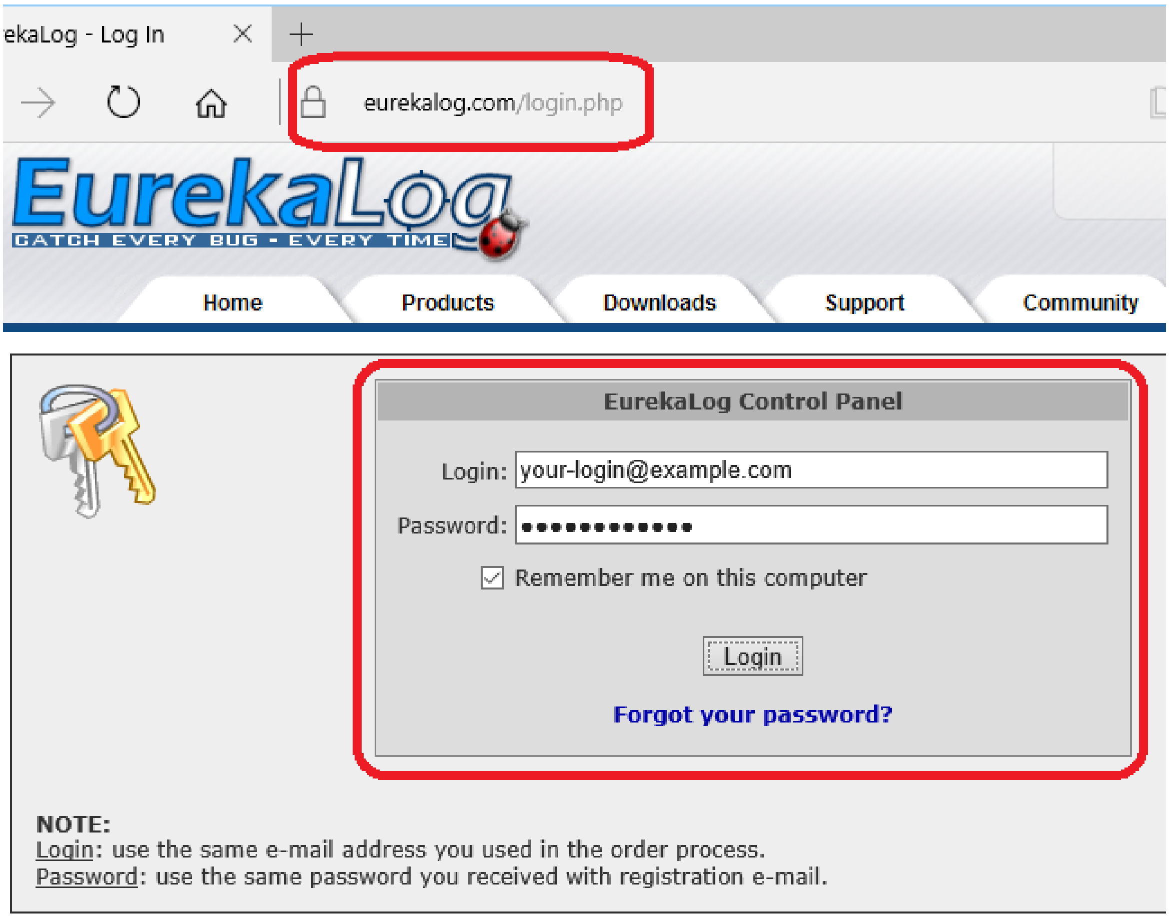
15.1.1 Upgrage guide

This short article will describe the process of upgrading to EurekaLog 7.

Obtaining EurekaLog 7

Upgrades from older versions (EurekaLog 6, EurekaLog 5, etc.) are sold with 50% discounts. To buy upgrade you should log in to your control panel.





EurekaLog Control Panel

Login: your-login@example.com

Password: ●●●●●●●●●●●●●

☑ Remember me on this computer

Login

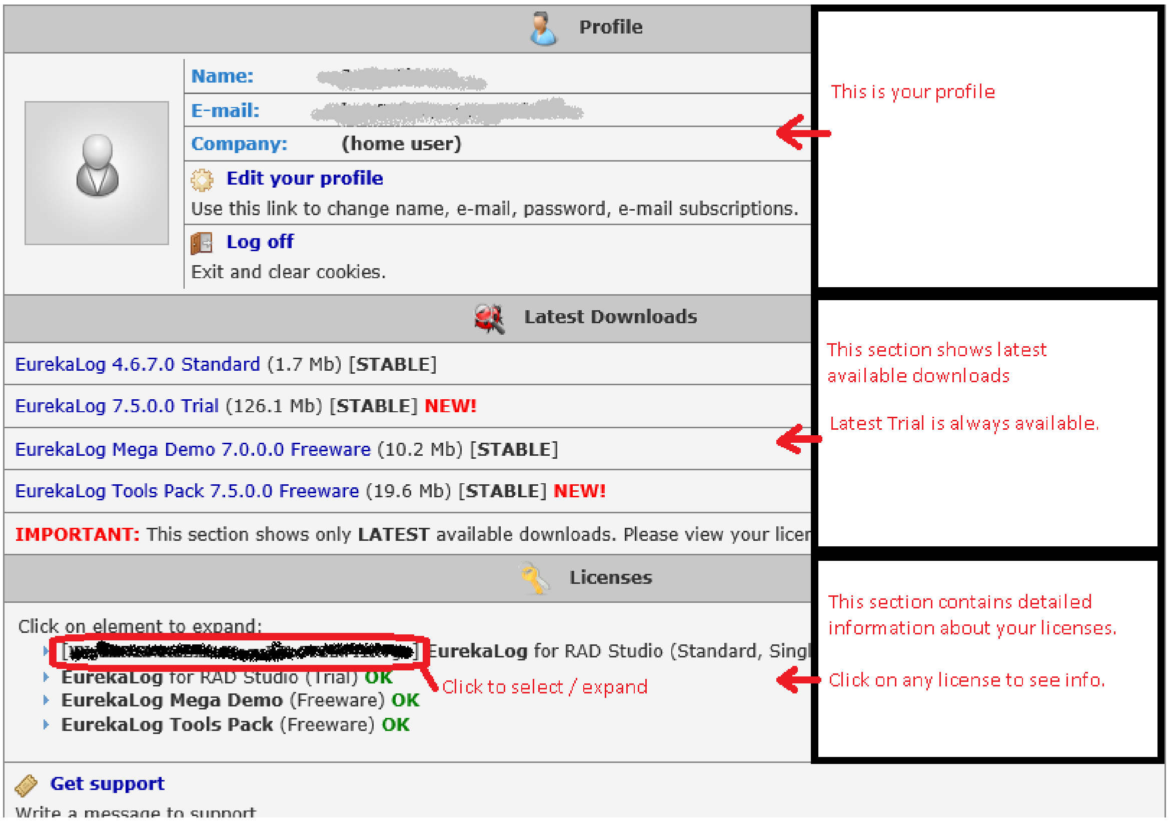
Forgot your password?

NOTE:
Login: use the same e-mail address you used in the order process.
Password: use the same password you received with registration e-mail.

Log in to your control panel

If you forget your password - you can recover it by using this page.

You will see your profile after successful log in:



Profile

Name: [redacted]

E-mail: [redacted]

Company: (home user)

Edit your profile

Use this link to change name, e-mail, password, e-mail subscriptions.

Log off

Exit and clear cookies.

This is your profile

Latest Downloads

EurekaLog 4.6.7.0 Standard (1.7 Mb) [STABLE]

EurekaLog 7.5.0.0 Trial (126.1 Mb) [STABLE] NEW!

EurekaLog Mega Demo 7.0.0.0 Freeware (10.2 Mb) [STABLE]

EurekaLog Tools Pack 7.5.0.0 Freeware (19.6 Mb) [STABLE] NEW!

IMPORTANT: This section shows only LATEST available downloads. Please view your licer

This section shows latest available downloads

Latest Trial is always available.

Licenses

Click on element to expand:

- [redacted] EurekaLog for RAD Studio (Standard, Singl
- EurekaLog for RAD Studio (Trial) OK
- EurekaLog Mega Demo (Freeware) OK
- EurekaLog Tools Pack (Freeware) OK

Click to select / expand

This section contains detailed information about your licenses.

Click on any license to see info.

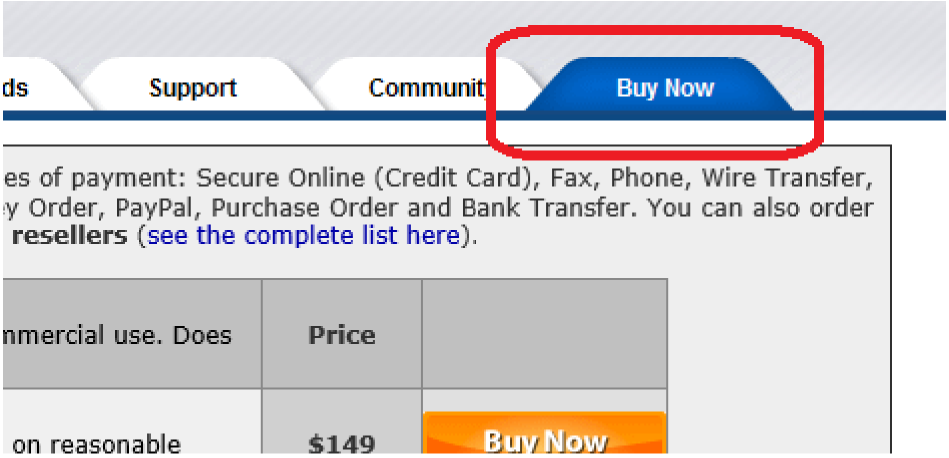
Get support

Write a message to support

User profile showing license info

There is EurekaLog 4 license in the example above. Therefore, there are download links to full version of EurekaLog 4 installer, as well as links to download latest free Trial, tools, demos.

Go to "Buy" page to buy upgrades:



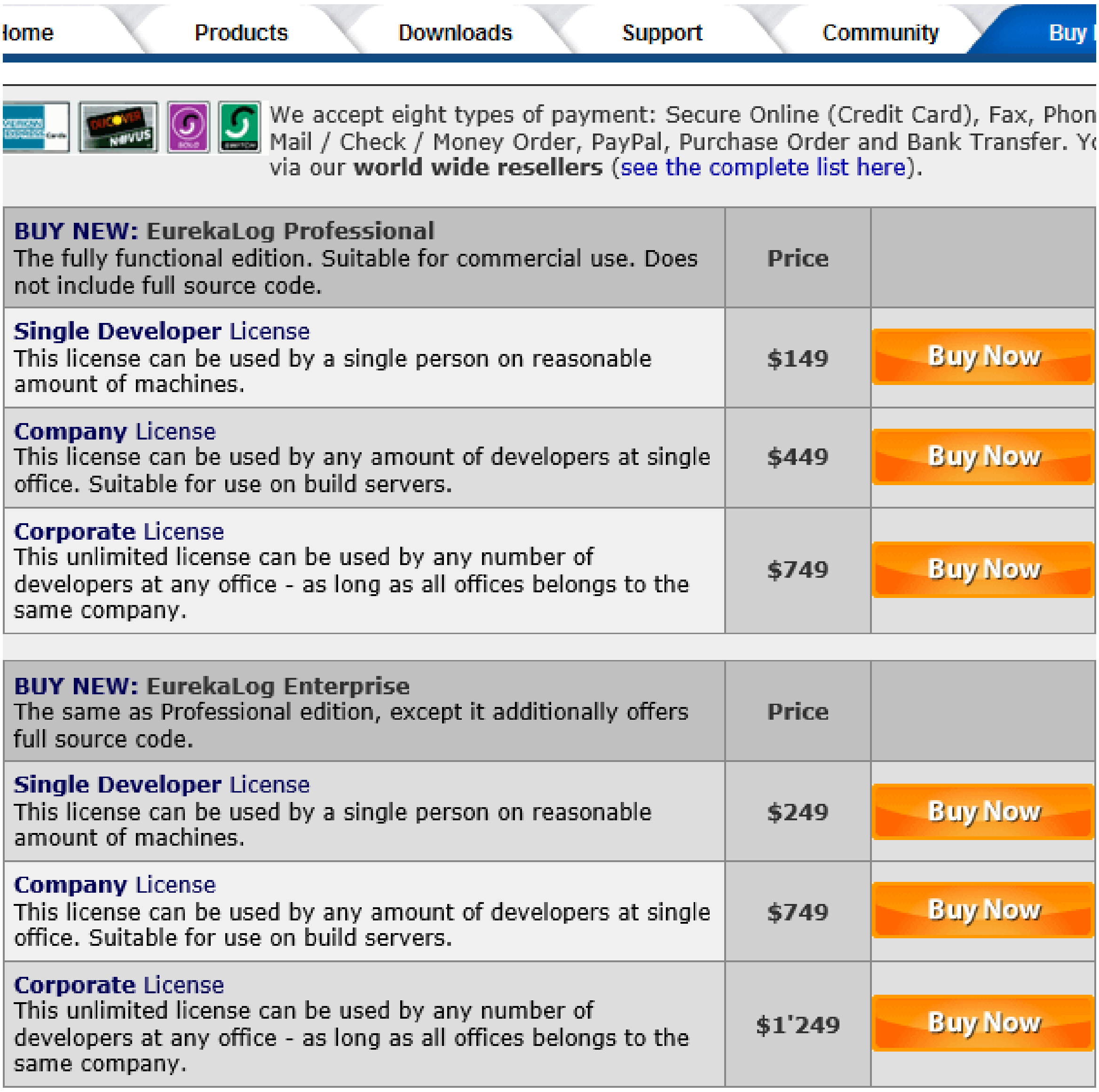
es of payment: Secure Online (Credit Card), Fax, Phone, Wire Transfer, y Order, PayPal, Purchase Order and Bank Transfer. You can also order resellers (see the complete list here).

nmercial use. Does	Price	
on reasonable	$149	Buy Now

Quick link to "Buy" page

Important: Please note that **you have to be logged in**. Otherwise you won't see upgrade options on "Buy" page. The content of the "Buy" page depends on your profile and licenses.

You will see two section on the "Buy" page. First section allows you to purchase a new license. For example, you may use this to purchase additional seats.



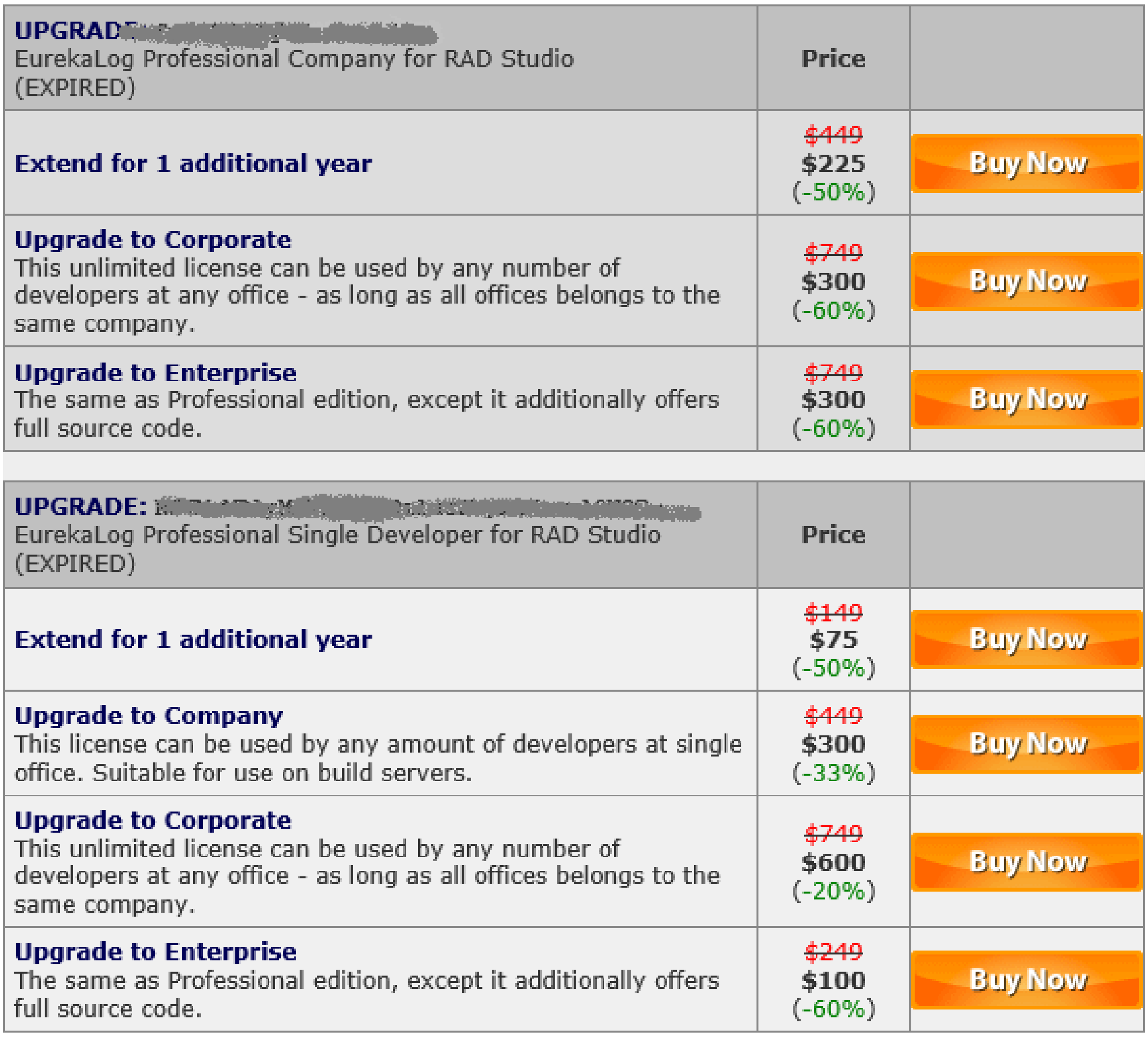
We accept eight types of payment: Secure Online (Credit Card), Fax, Phon
Mail / Check / Money Order, PayPal, Purchase Order and Bank Transfer. Yo
via our **world wide resellers** (see the complete list here).

BUY NEW: **EurekaLog Professional** The fully functional edition. Suitable for commercial use. Does not include full source code.	**Price**	
Single Developer License This license can be used by a single person on reasonable amount of machines.	**$149**	Buy Now
Company License This license can be used by any amount of developers at single office. Suitable for use on build servers.	**$449**	Buy Now
Corporate License This unlimited license can be used by any number of developers at any office - as long as all offices belongs to the same company.	**$749**	Buy Now

BUY NEW: **EurekaLog Enterprise** The same as Professional edition, except it additionally offers full source code.	**Price**	
Single Developer License This license can be used by a single person on reasonable amount of machines.	**$249**	Buy Now
Company License This license can be used by any amount of developers at single office. Suitable for use on build servers.	**$749**	Buy Now
Corporate License This unlimited license can be used by any number of developers at any office - as long as all offices belongs to the same company.	**$1'249**	Buy Now

Top of "Buy" page is always the same -
it allows you to buy a new license

Next, the second section will show all available upgrade options. Scroll down to see these discounted options:



UPGRAD[redacted] EurekaLog Professional Company for RAD Studio (EXPIRED)	Price	
Extend for 1 additional year	~~$449~~ **$225** (-50%)	Buy Now
Upgrade to Corporate This unlimited license can be used by any number of developers at any office - as long as all offices belongs to the same company.	~~$749~~ **$300** (-60%)	Buy Now
Upgrade to Enterprise The same as Professional edition, except it additionally offers full source code.	~~$749~~ **$300** (-60%)	Buy Now

UPGRADE: [redacted] EurekaLog Professional Single Developer for RAD Studio (EXPIRED)	Price	
Extend for 1 additional year	~~$149~~ **$75** (-50%)	Buy Now
Upgrade to Company This license can be used by any amount of developers at single office. Suitable for use on build servers.	~~$449~~ **$300** (-33%)	Buy Now
Upgrade to Corporate This unlimited license can be used by any number of developers at any office - as long as all offices belongs to the same company.	~~$749~~ **$600** (-20%)	Buy Now
Upgrade to Enterprise The same as Professional edition, except it additionally offers full source code.	~~$249~~ **$100** (-60%)	Buy Now

Discounted prices on "Buy" page

Important: Discounted options are only shown for logged in customers. **You won't be able to see discounted prices if you are not logged in.**

There will be a block of options for each of your licenses. Use "Extend for 1 additional year" option to upgrade your old license to latest EurekaLog version.

You will also be able to buy upgrade within your old version (for example, upgrade EurekaLog 6 Professional to EurekaLog 6 Enterprise, or upgrade Single Developer to Company) - for just a price difference between editions.

If you can't use our direct "Buy" page - then you can use any of our world-wide resellers. Pick a reseller and ask it to purchasing upgrade for your EurekaLog license.

Once update is obtained - you will see download links to full version of latest EurekaLog installer in your control panel. Be sure to use the correct login/password pair that is sent to you in **order notification e-mail**. Make sure this e-mail is not blocked. Check your spam folder.

If you have any problems with buying/upgrading/licenses - ask us (sales department).

Installing EurekaLog 7 over EurekaLog 6

You don't need to uninstall EurekaLog 6 before installing EurekaLog 7. Both versions could be installed on the same machine. You can switch between EurekaLog versions by using "Start Menu" / "Programs" / "EurekaLog 6" / "Manage EurekaLog in IDEs" (this option will appear after installation of EurekaLog 7) and "Start Menu" / "Programs" / "EurekaLog 7" / "Manage EurekaLog in IDEs" menu item (make sure to run both tools under administrator account).

Use corresponding "Manage" tool to enable/disabled EurekaLog for supported IDEs. For example, if you have EurekaLog 7 installed for Delphi 7 and want to use EurekaLog 6:

1. Run EurekaLog 7 "Manage" tool:
   a. Select Delphi 7 IDE;
   b. Click on "Remove EurekaLog";
2. Run EurekaLog 6 "Manage" tool:
   a. Select Delphi 7 IDE;
   b. Click on "Install EurekaLog 6 (recommended)".

If you want to switch back from EurekaLog 6 to EurekaLog 7:

1. Run EurekaLog 6 "Manage" tool:
   a. Select Delphi 7 IDE;
   b. Click on "Remove EurekaLog";
2. Run EurekaLog 7 "Manage" tool:
   a. Select Delphi 7 IDE;
   b. Click on "Install EurekaLog 7 (recommended)".

Be sure to close all running IDE instances when installing/uninstalling EurekaLog or using "Manage" tool.

Of course, you can fully uninstall EurekaLog 6 and then install EurekaLog 7. However, you must be sure that your project will work. If you haven't tried this yet - then abandoning EurekaLog 6 is not a wise move.

Possible installation issues

If you do not see EurekaLog 7 in your IDE - please use "Start Menu" / "Programs" / "EurekaLog 7" / "Manage EurekaLog in IDEs" menu item to activate EurekaLog (make sure to run it under administrator account).

For other issues - please see our automatic troubleshooter or troubleshooting guide 596.

To diagnose installation problems - please run C:\Program Files (x86)\Neos Eureka S.r.l \EurekaLog 7\Bin\InstallationDiagnostic.exe tool to generate log file. You can send this log to us. Additionally you may attach installation logs which can be found at: C:\Program Files (x86)\Neos Eureka S.r.l\EurekaLog 7\InstallationDiagnostic.log and Process.log.

Upgrading your project to EurekaLog 7

EurekaLog 7 contains A LOT of changes from EurekaLog 6. We beta-tested EurekaLog 7 for 1.5 years - about half year of public beta testing and 17 months of private beta-testing. And we done much to ensure that upgrade process will be as much painless as possible. However, we surely do not expect that upgrade process will go smooth. There always be problems. There are tons of real-life live EurekaLog-enabled projects around the world, and we just can not test them all.

1. Therefore, **the first thing that you need to do - is to backup your project before opening it in IDE with EurekaLog 7 installed**. Typically, you only need to backup your project files (i.e. like Project1.*). Then you can open your project in EurekaLog 7.
2. Once you have opened your project - please, review its options 234 on each page. Make sure options are set to expected values. Alternatively, you may want to "Reset" options to defaults and make customization changes.
3. Try to compile your project. There may be errors due to changes in EurekaLog's code. The list of changes is available here 624. Alter your code according to this guide.

Compilation problems

If you encounter errors like "*File XYZ was compiled with different version of ABC*" or "*Could not compile used unit XYZ*" - then your project uses mix of old and new files.

- **Be sure to clear your project from old .dcu, .obj and other generated files.**

- Make sure that your project do not contain explicit search paths to other EurekaLog version.
- Make sure that your IDE do not contain explicit search paths to other EurekaLog version.
- Try to run a disk-wide file search for EurekaLog and your project files. See if there are any unexpected duplicates.
- Sometimes it's possible that IDE do not update changes in settings. Try to run Start/Programs/EurekaLog 7/Manage tool and click on "No EurekaLog" and then - on "EurekaLog 7 with IDE expert" (make sure you close IDE before doing that and make sure to run the Manage tool under administrator account). If this will not help - try to (re)setup search paths for your IDE as explained here 604.

Please see also this article 598. Also, try troubleshooter and check common problems in our Knowledge Base.

Get familiar with changes in EurekaLog 7

EurekaLog 7 has many changes. Some old ways of doing things were changed. Some are considered obsolete and replaced with other methods.

You can find demos under "Start Menu" / "Programs" / "EurekaLog 7" / "Demos" menu item.
You can find quick-start guides here 45.
You can watch video tutorials here.
You can read migration reference here 624.

Get support with upgrading

If you can't resolve some issue with upgrading - then just ask us (tech-support department).

15.2 5.x -> 6.x

These are the changes from the old 5.x version to the new 6.0 version:

1. CustomDataRequest parameter event changed ("CustomData: String" parameter is replaced with the new "DataFields: TStrings")
2. CustomFieldsRequest event removed (replaced with the new "CustomWebFieldsRequest" event)
3. FreezeMessage property removed (replaced with a "Messages Text Tab" field)
4. ExceptionClassName property removed (replaced with the new "ExceptionsFilters")
5. ExceptionMessage property removed (replaced with the new "ExceptionsFilters")
6. ShowTerminateBtn property removed (replaced with the "TerminateBtnOperation = tbnone" value)
7. TExceptionDialogOption.edoShowExceptionDialog replaced with TExceptionDialogType
8. TExceptionDialogOption.edoSendEmailChecked replaced with TExceptionDialogType.edoSendErrorReportChecked
9. TCommonSendOption.sndCompressAllFiles removed
10. TBehaviourOption.boActivateCrashDetection replaced with AutoCrashOperation
11. TLogOption.loSaveModulesSection replaced with TLogOption.loSaveModulesAndProcessesSections
12. TLogOption.loSaveCPUSection replaced with TLogOption.loSaveAssemblerAndCPUSections

That's all.

See also:

- Migration from 6.x -> 7.x 624
- What's new in EurekaLog 6.0 634
- What's new in EurekaLog 6.1 634
- Old documentation

15.2.1 What's New in EurekaLog 6.0

Caution: this is old EurekaLog 6 documentation. This is not the latest version.

EurekaLog 6 includes new features and enhancements in the following areas:

New IDE features:

- The EurekaLog Viewer is more similar to a stand alone BUG tracking tool
- Full revisited EurekaLog Options form
- New "IDE/Tools/EurekaLog IDE Settings" form
- Run custom programs before and after every project build
- History integration (for the unit line searching on modified sources)
- Options to reduce the .EXE file size
- Option to detect .EXE cracks

New application capabilities:

- Catches of every MEMORY-LEAKS (see the "Memory Leaks Limit" page) !!!
- Delivery of every BUG to the most used Web BUG-Tracking tools
- Display the Dis-Assembler section
- Display the Processes list section
- Display of more Hardware and Software info (DPI, printer, VGA, privileges, ...)
- Full .jdbg (Jedi Debug file), .MAP and .TDS (TD32 Debug file) support
- Full customizable Exception-Dialog (with more styles - as MS style)
- Full customizables Exceptions Filters (can choose style, behavior, messages, ...)
- All the files to send are compresses and encrypted in ZIP format
- Full COM and SafeCall Exceptions customization
- Optionally catches every HANDLED EXCEPTION!!!
- Fully customizable message texts collections (for multi-language applications)
- Environment variables (%EnvironmentVariablet%) support
- New ZIP compress file format to (password encryption allowed)
- Add a customizable HELP button (call an event)
- Full UNICODE logs handling

...and much, much more!

Compatibility issues:

- For any "Compatibility Issues" try to see the "Changed from the old 5.x version" 633 section!

Features list:

- For a more detailed features list, see the "Features" page.

15.2.2 What's New in EurekaLog 6.1

Caution: this is old EurekaLog 6 documentation. This is not the latest version.

Q: What is new in EurekaLog 6.1?
A: EurekaLog 6.1 is exactly the same as EurekaLog 6.0 (6.0.25 is the latest available version), except one "little" detail: we've added support for new compiler from EurekaLog 7.0 (which is in development right now).

Q: Why is it 6.1? Why not 6.0.26?
A: We do only bug fixes in 6.x branch, so we get 6.0.10, 6.0.20, 6.0.25, etc. Each new version (like 6.0.xx) contains bug fixes, but no new features. Since we've added a major new feature - we decided to increase a version number, so new version became 6.1 -

indicating addition of a major new feature.

Q: Is it safe to upgrade? Am I affected?
A: Don't worry, 6.1.xx behaves as 6.0.xx by default. You can safely upgrade and no new changes will affect you (except the usual bug fixes). New compiler have no effect, until explicty enabled.

Q: What's so great about new compiler?
A: In a recent 3 months there were many reports about problems with EurekaLog and debug information (i.e. mapping between code and human-readable names of units, methods, etc). After investigation we've discovered that there appears linker bugs (usually on large projects), which prevents EurekaLog from proper functioning.

What does it mean? It means that for some large projects EurekaLog can't show you a proper call stack because of wrong debug information, which is usually caused by linker's bugs.

OK, so the new compiler from EurekaLog 7 have a new architecture and a new code (it's re-written almost on 80%) and it is able to solve these problems (along with help of new format of debug information).

Another advantages are:

1. Increased speed. New compiler should be (theoretically) faster than old, since we've reduced amount of work with strings.
2. Reduced memory consumption. New compiler should be (theoretically) more conservative about memory usage, since we're working on mapped file, instead of parsing it into separate strings.
3. Increased stability. Any issues with compiler itself does not affect IDE, since compiler is now run into separate process.
4. Increased detalization. New compiler is able to store routines without line numbers. Old compiler can't do this. Usually this doesn't matter anyway, since we always recommend to enable "Use Debug DCUs" option.

Items 2 and 3 allows you to use EurekaLog for very large projects, which wasn't possible before (because of shared virtual address space with IDE and, thus, hitting a memory limits).

Word "theoretically" in items 1 and 2 means that "we think so", but we don't have any exact estimates yet. So, if you're owning a large project (say, 10+ MB .exe file or 15+ MB .map file) - please, send us (even very approximate) details about how fast (or slow!) compilation is with old and new compiler.

Q: Why add the compiler from EurekaLog 7 to EurekaLog 6? Why not just fix issues in EurekaLog 6? Or just wait for EurekaLog 7 release?
A: Because it's impossible to fix these issues with old logic of compiler (there are much technical details here, I won't going to discuss them). To solve these issues in EurekaLog 6, one needs to almost re-write the compiler - pretty much the same job as was already completed in EurekaLog 7 (compiler upgrade in EurekaLog 7 wasn't because of these issues - we just improved code. Solving these issues was a side-effect).

So why just not port code from EurekaLog 7 to EurekaLog 6? Well, we tried this, but it's not easy too, since there are a lot of changes (also remember that we dropped support for Delphi 3, so we used new features a lot).

Why not just leave this as is, claiming that customers should wait for EurekaLog 7? I'm not saying that EurekaLog 7 have a long road before release, but I'm saying that we care about our customers, so we've decided to release intermediate EurekaLog 6.1, which solves some major problems of our customers.

Thus, the only solution left is to include new compiler as an option to EurekaLog 6.

Q: When I should use new compiler?
A: If you have small/medium project and never suffer from weird call stacks - then you can just ignore it. Don't use new compiler (which is default, BTW). Otherwise - try it. New

compiler may help you to solve your issues.

Oh, and one more thing - this compiler doesn't do a lot for C++ Builder. That's because .map files of C++ Builder miss line numbers, so EurekaLog uses .tds file instead. In other words, there is no improvement on debug information for C++ Builder.

Q: How to use it?
A: Very simple. Just go to Project / EurekaLog options, select Build options tab and enable Use EurekaLog 7 compiler checkbox. See also.

Q: What about command line compilation and build automation?
A: That is possible too. You can find new compiler in your EurekaLog folder under sub-folder which corresponds to your IDE. For example:

C:\Program Files\Neos Eureka S.r.l\EurekaLog 6\Delphi15\el7c.exe

The same .exe file can be used for both Delphi and C++ Builder - just be sure that you pick a correct version (in case if you have several different versions of IDE installed).

You can use this file exactly as the old ecc32.exe/emake.exe. Except there are few additional options - see command line options for more information.

Q: I don't want to use experimental compiler from a EurekaLog 7, which doesn't even have a stable release!
A: First I want to highlight that EurekaLog's code remain the same. New compiler doesn't affect the code. It's still old good and tested 6.0.25. New compiler affects only additional debug information. So, if you have problems with debug information, new compiler may solve it. I.e. it's not worse than it's already is. Surely, you may not use new compiler, if you don't need to.

Yes, new compiler comes from unstable version of EurekaLog 7. However, we've tested a new compiler quite strict. Sure we may miss something, but it's quite stable now.

Of course, you shouldn't just enable this option and throw your application to your customers. You need to do a few simple tests:

- Try just to compile your application with new compiler - will it be successfull? New compiler contains additional asserts, which trigger on unsupported cases. Usually this means that your project contains new case which we doesn't cover yet.
- Take a known issue with a bug report and check this issue against new compiler - will a new bug report be better than old?
- You can place a button, which raises exception somewhere deep inside your code. Will a bug report be detailed enough?

If these tests will pass - then you can safely use new compiler. If not - please, contact us, so we can improve our product! Just send us a your .map file with description of the problem.

15.3 4.5.x -> 5.x

These are the changes from the old 4.5.x version to the new 5.0 version:

1. SMTPShowDialog property removed (replaced by CommonSendOptions.sndShowSendDialog)
2. SendEntireLog property removed (replaced by CommonSendOptions.sndSendEntireLog)
3. EurekaLogLook property removed (replaced by ExceptionDialogOptions.edoUseEurekaLogStyle)
4. ShowExceptionDialog property removed (replaced by ExceptionDialogOptions.edoShowExceptionDialog)
5. EmailSendOptions property renamed in EmailSendMode (with suffix replaced from 'eso' to 'esm')
6. TEurekaExtraInformation.CompiledDate replaced with TEurekaExtraInformation.CompilationDate
7. Added the 'so' prefix at every TShowOption items

That's all.

See also:
- Migration from 5.x -> 6.x 633
- Migration from 6.x -> 7.x 624

15.4 4.x -> 4.5.x

These are the changes from the old 4.x version to the new 4.5 version:

1. Changed the EmailObject property in EmailSubject
2. Changed the TEmailSendOptions type to "(esoNoSend, esoEmailClient, esoSMTPClient, esoSMTPServer)"
3. Changed the TLogOption type to "(loNoDuplicateErrors, loAppendReproduceText)"
4. Changed the AppendToLog property to AppendLogs
5. Changed the "MuteMode" property to "ShowExceptionDialog" (with inverted sense)
6. Changed the TForegroundType
7. Changed the TMessageType type
8. Changed the TShowOption type
9. EMailSendConfirm property removed
10. EResFile.pas removed

That's all.

See also:
- Migration from 4.5.x -> 5.x 636
- Migration from 5.x -> 6.x 633
- Migration from 6.x -> 7.x 624

15.5 3.x -> 4.x

Please read the following instructions if you wish to install EurekaLog 4.x as an upgrade from EurekaLog 3.x:

1. Check the new "Exception Log Options". Some of the old options are no longer available. Save your old options before continuing.
2. Replace the old "ExceptionHandle.OnException" event with the new ExceptionNotify event.
3. Don't use the "TEurekaThread.EurekaHandleException" method for managing thread exceptions. EurekaLog now has fully-automatic support for multithreaded applications.
4. Change all references to the ExceptionLog2 unit to ExceptionLog. The ExceptionLog2 unit was used in the old version for managing Console applications.

That's all.

See also:
- Migration from 4.x -> 4.5.x 637
- Migration from 4.5.x -> 5.x 636
- Migration from 5.x -> 6.x 633
- Migration from 6.x -> 7.x 624

Part XVI

16 License

EurekaLog Software License and Limited Warranty.

Before proceeding with the installation and/or use of this software, carefully read the following terms and conditions of this license agreement and limited warranty. By installing or using this software you indicate your acceptance of this agreement. If you do not accept or agree with these terms, you may not install or use this software!

This software, including source code, documentation, compiled code and all additional materials (the "Software") is owned by Fabio Dell'Aria.

This Software is protected by copyright laws. At all times the Software author retains full title to the software. Subject to your acceptance of and accordance with the terms and conditions stated in this agreement, you shall be granted a software license.

The author hereby grant you a non-exclusive, royalty free license to use the Software as set forth below:

1....integrate the Software with your Applications, subject to the redistribution terms below;
2....modify or adapt the Software in whole or in part for the development of Applications based on the Software.

Only for **single** license:
- You may install a copy of the Software on a computer Desktop and/or a Notebook (without simultaneous use) and freely move the Software from one computer to another, provided that you are the only individual using the Software. If you are an entity, you must designate one individual within your organization to have the right to use the Software.

Only for **company** license:
- Every developer of the company (limited to only one geographical address) that has bought this license, can install and use the software.

Only for **corporate** license:
- Every developer of the company (without any limits about the geographical address) that has bought this license, can install and use the software.

REDISTRIBUTION RIGHTS
You are granted a non-exclusive, royalty-free right to reproduce and redistribute executable files created using the Software .

RESTRICTIONS
Without the expressed, written consent of Software author, you may NOT:

1....distribute the Software source code or modified versions;
2....rent, lease or sell any portion of the Software on its own, without integrating it into your executable files.

TRIAL VERSION
The Software Trial version may be freely distributed and/or used with
exceptions noted below, provided the Software is not modified in any way.

1....No person or company may distribute/uses separate parts of the Software Trial version without written permission of the author;
2....The Software Trial version may not be distributed/uses inside of any other software package without written permission of the author;
3....Hacks/crack, keys or key generators may not be uses/distributed.

CHANGES TO SOURCE CODE

The Software author reserves the right to change any part of the source in future versions of the product. These changes may include the removal of classes, properties and methods or the creation of new classes, properties and methods.

SELECTION AND USE
You assume full responsibility for the selection of the Software to achieve your intended results and for the installation, use and results obtained from the Software.

LIMITED WARRANTY
This software is provided "as is" without warranty of any kind either expressed or implied, including but not limited to the implied warranties merchantibility and fitness for a particular purpose. The entire risk as to the quality and performance of the product is with you. Should the product prove defective, you assume the cost of all necessary servicing or error correction. Author do not warrants that the functions contained in the software will meet your requirements or that the operation of the software will be uninterrupted or error free.

LIMITATION OF REMEDIES AND LIABILITY.
In no event shall Software author, or any other party who may have distributed the software as permitted above, be liable for damages, including any general, special, incidental, or consequential damages arising out of the use or inability to use the software (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or failure of the software to operate with any other products), even if such holder or other party has been advised of the possibility of such damages.

Index

- % -

%_BugID% 413
%_BugReport% 413
%_CallStack% 413
%_ExceptModuleDesc% 413
%_ExceptModuleName% 413
%_ExceptModulePath% 413
%_ExceptModuleVer% 413
%_ExceptMsg% 413
%_ExceptType% 413
%_IDEConfig% 413
%_IDEDst% 413
%_IDEProject% 413
%_IDESource% 413
%_IDESrc% 413
%_IDETarget% 413
%_LineBreak% 413
%_MainModuleDesc% 413
%_MainModuleName% 413
%_MainModulePath% 413
%_MainModuleVer% 413
%_Reproduce% 413
%_ThisModuleDesc% 413
%_ThisModuleName% 413
%_ThisModulePath% 413
%_ThisModuleVer% 413
%BUG_ID% 296
%BUG_REPORT% 296
%CALL_STACK% 296
%CONTENT_TYPE% 296
%EXCEPTION_CLASS% 296
%EXCEPTION_LOCATION% 296
%EXCEPTION_MESSAGE% 296
%FILE_NAME% 296
%TITLE% 296

- . -

.dbg 412
.eof 439, 443, 445, 448, 449, 450
.jdbg 412
.pdb 412
.tds 411

- A -

ActivateFilters 343
administrator 31
application type 45
Armadillo 520
ASPack 520
ASProtect 520
AsyncCalls 565
AttachedFiles 304
attributes 185, 190
AutoCloseDialogSecs 271, 279
AutoCrashMinutes 259
AutoCrashNumber 259
AutoCrashOperation 259

- B -

basic procedures 20, 33, 38, 40, 45, 46, 52, 53, 55, 58, 68
boCopyLogInCaseOfError 341
boDoNotPauseELServiceThread 246
boDoNotPauseMainThread 246
boPauseELThreads 246
boPauseRTLThreads 246
boPauseWindowsThreads 246
boSaveCompressedCopyInCaseOfError 341
BoxedApp 520
breakpoints 614
bug 40
bug report 40, 46
bug tracker send 68, 404, 406, 407, 408
BugZilla 53, 55, 68, 105, 134, 331, 407
buy 14

- C -

C++ Builder 58
call stack 40, 79, 81, 83, 95, 96, 97, 246
CExe 520
chained exception 244, 573
changes 624, 633, 636, 637
changes from the old 3.x version 637
changes from the old 4.5.x version 636
changes from the old 4.x version 637
changes from the old 5.x version 633
changes from the old 6.x version 624
CodeVirtualizer 520
compatibility 624, 633, 636, 637
compilation 421, 423, 424, 426, 429, 431, 432
coXYZ options 266

cryptor 520
csoAllowedRenderMethods 237
csoCaptureDelphiExceptions 237
csoCaptureOnlyModuleExceptions 237
csoShowBPLs 237
csoShowDLLs 237
csoShowELThreads 246
csoShowInvalid 237
csoShowPointers 237
csoShowRTLThreads 246
csoShowWindowsThreads 246
custom attributes 185, 190
CustomExceptionMessage 244
customization 45, 46, 52, 53, 180, 183, 185, 189, 190, 192, 195, 211, 212

- D -

dbg 412, 496, 527
debug information 38, 40, 58, 243, 409
debug paths tool 617
DEBUG_EL_CODE 613, 619, 620
debugging 70, 614
Delphi 58
dialog 52
dlgMsgBoxUseNative 268
DoNotTouch 448, 449
DoNotTouchCompilation 448
DoNotTouchOptions 448, 449
DoNotTouchPackages 448, 449
DoNotTouchUnits 448, 449

- E -

EDD 68
edoAttachScreenshotChecked 279
edoMandatoryEMail 271
edoOwnedWindow 271, 279
edoRestartChecked 271, 279
edoSendErrorReportChecked 268, 271, 279, 292
edoShowAttachScreenshotOption 279
edoShowCopyToClipOption 279
edoShowCustomButton 271, 279
edoShowDetailsButton 271, 279
edoShowEMailControl 271
edoShowInTopMostMode 271, 279
edoShowModal 271, 279
edoShowSendErrorReportOption 268, 271, 279, 292
edoUseEurekaLogLookAndFeel 279
edoUseRealIcon 271, 279
edoUseRealName 271, 279
edtConsole 292
edtConsoleDetailed 292
edtConsoleDump 292
edtEurekaLog 279
edtEurekaLogDetailed 279
edtMessageBox 268
edtMessageBoxDetailed 268
edtMSClassic 271
edtNone 267
edtService 295
edtWEB 296
edtWER 300
el_debug 613
--el_debug 613
el_debug_standalone 613
--el_debug_standalone 613
e-mail send 391, 393, 396, 397, 398
enable EurekaLog in your project 33, 606
Enigma 520
error 40
ErrorsNumberToSave 264
ErrorsNumberToShowTerminateBtn 271, 279
esmMAPI 315
esmShellClient 309
esmSimpleMAPI 314
esmSMTPClient 316
esmSMTPServer 319
EurekaLog 2, 4, 8, 16, 40, 68, 70, 180
EurekaLog 6 624, 627, 633, 634
EurekaLog 7 2, 4, 624, 627
EurekaLog debug paths tool 617
EurekaLog integral parts 221
EurekaLog module informer tool 617
EurekaLog parts 221
EurekaLog PE analyzer tool 617
EurekaLog settings editor tool 617
EurekaLog threads snapshot tool 617
EurekaLog viewer tool 617
EventLogCategory 295
EventLogComputer 295
EventLogEventID 295
EventLogName 295
exception 40
exception chained 40, 244, 573
exception driven development 68
exception filter 343, 344
exception filters 185, 190
exception handler 40
exception log 40
exception nested 40, 244, 573
exception tracer 40
ExceptionDialogType 267
ExceptionsFilters 343

EXE Bundle 520
exe cryptor 520
exe packer 520
exe protector 520
EXE Stealth 520
executable module 40
eXPressor 520
extended MAPI 396
external settings editor tool 617

- F -

FAQ 215, 217, 218
features 10
feedback 105
file 217, 218
file formats 218
file saving 46, 217, 218
files 217, 218
filter 343, 344
filters 185, 190
FogBugz 53, 55, 68, 105, 108, 115, 322, 404
ForegroundTab 279
formats 218
FreezeDisableUnderDebugger 257
FSG 520
FTP upload 321, 404

- H -

handled exception 40, 70
hardware exception 40
hook 40
hooking 40
how to configure bug report 46
how to configure dialog 52
how to configure send 53
how to enable EurekaLog in your project 33
how to select application type 45
how to use EurekaLog 33, 38
HTMLLayout 296
HTTP upload 320, 398

- I -

injecting 40
installation 20, 31, 596
integral parts 221
interface 222
internal errors 591

- J -

JCL 412, 495, 526
JIRA 53, 55, 68, 105, 143, 335, 408

- L -

leak 40, 70, 97
license 10, 14, 639
limited accounts 31
loAddBugIDInLogFileName 264
loAddComputernameInLogFileName 264
loAddDateInLogFileName 264
loAppendReproduceText 271, 279
loAppendReproduceTextOnlyWhenSending 279
loDeleteLogAtVersionChange 264
log 40, 218
log formats 218
log report 40
loNoDuplicateErrors 264
loSaveAssemblerAndCPUSections 266
loSaveModulesSection 266
loSaveProcessesSection 266

- M -

madExcept 413, 495, 526
mailto: 309, 391
Mantis 53, 55, 68, 105, 119, 327, 406
MAPI 315, 393, 396
memory leak 40, 70, 97
MEW 520
Microsoft 412
microsoft visual studio 496, 527
migration 624, 627, 633, 636, 637
migration from the old 3.x version 637
migration from the old 4.5.x version 636
migration from the old 4.x version 637
migration from the old 5.x version 633
migration from the old 6.x version 624, 627
module 40, 617
module informer tool 617
MPRESS 520
multi-threading 246

- N -

nested exception 244, 573
NestedExceptionMessage 244
NestedExceptionStack 244

non-administrator 31

- O -

Obsidium 520
OmniThreadLibrary 565
OutputPath 264

- P -

packer 520
panic mode 591
parts 221
password 243
pause 246
pdb 412, 496, 527
PE analyzer tool 617
PELock 520
PESpin 520
post-processing 38
problems 72, 596, 606, 614, 615
project 33, 38, 45, 58
project groups 450
protector 520

- Q -

quick start 20, 33, 38, 40

- R -

report 40, 218
report formats 218
reporting 68
resource leak 40, 70, 97
resume 246
RLPack 520

- S -

SaveLogFile 264
saving bug report to file 46
saving to file 46
selecting application type 45
selecting dialog 52
selecting send 53
send, bug tracker 404, 406, 407, 408
send, BugZilla 407
send, comparison 55
send, e-mail 391, 393, 396, 397, 398
send, extended MAPI 396
send, FogBugz 404
send, FTP upload 404
send, HTTP upload 398
send, JIRA 408
send, mailto 391
send, Mantis 406
send, MAPI 393, 396
send, overview 68, 390
send, shell 391
send, simple MAPI 393
send, SMTP client 397
send, SMTP server 398
send, web 398, 404, 406, 407, 408
SendBugZillaAppendText 331
SendBugZillaBasicAuthLogin 331
SendBugZillaBasicAuthPassword 331
SendBugZillaComponent 331
SendBugZillaCountFieldName 331
SendBugZillaLogin 331
SendBugZillaOwner 331
SendBugZillaPassword 331
SendBugZillaPort 331
SendBugZillaProject 331
SendBugZillaProxyHost 331
SendBugZillaProxyLogin 331
SendBugZillaProxyPassword 331
SendBugZillaProxyPort 331
SendBugZillaSSL 331
SendBugZillaUploadFilesForDups 331
SendBugZillaURL 331
SendBugZillaUseHardware 331
SendBugZillaUseOS 331
SendBugZillaUseVersion 331
SenderClasses 302
SendFogBugzAppendText 322
SendFogBugzArea 322
SendFogBugzBasicAuthLogin 322
SendFogBugzBasicAuthPassword 322
SendFogBugzCategory 322
SendFogBugzLogin 322
SendFogBugzOwner 322
SendFogBugzPassword 322
SendFogBugzPort 322
SendFogBugzProject 322
SendFogBugzProxyHost 322
SendFogBugzProxyLogin 322
SendFogBugzProxyPassword 322
SendFogBugzProxyPort 322
SendFogBugzSSL 322
SendFogBugzUploadFilesForDups 322
SendFogBugzURL 322
SendFogBugzUseComputer 322
SendFogBugzUseEMail 322

SendFogBugzUseVersion 322
SendFTPLogin 321
SendFTPPassiveMode 321
SendFTPPort 321
SendFTPProxyHost 321
SendFTPProxyLogin 321
SendFTPProxyPassword 321
SendFTPProxyPort 321
SendFTPURL 321
SendHTTPAuthLogin 320
SendHTTPAuthPassword 320
SendHTTPPort 320
SendHTTPProxyHost 320
SendHTTPProxyLogin 320
SendHTTPProxyPassword 320
SendHTTPProxyPort 320
SendHTTPSSL 320
SendHTTPURL 320
SendJIRAAppendText 335
SendJIRAComponent 335
SendJIRACountFieldName 335
SendJIRAIssueType 335
SendJIRALogin 335
SendJIRAOwner 335
SendJIRAPassword 335
SendJIRAPort 335
SendJIRAProject 335
SendJIRAProxyHost 335
SendJIRAProxyLogin 335
SendJIRAProxyPassword 335
SendJIRAProxyPort 335
SendJIRASSL 335
SendJIRAUploadFilesForDups 335
SendJIRAURL 335
SendMantisAppendText 327
SendMantisBasicAuthLogin 327
SendMantisBasicAuthPassword 327
SendMantisCategory 327
SendMantisCountFieldName 327
SendMantisLogin 327
SendMantisOwner 327
SendMantisPassword 327
SendMantisPort 327
SendMantisProject 327
SendMantisProxyHost 327
SendMantisProxyLogin 327
SendMantisProxyPassword 327
SendMantisProxyPort 327
SendMantisSSL 327
SendMantisUploadFilesForDups 327
SendMantisURL 327
SendMAPIAppendLogs 315
SendMAPIMessage 315
SendMAPISubject 315
SendMAPITarget 315
SendShellEncode 309
SendShellMessage 309
SendShellSubject 309
SendShellTarget 309
SendShellUseShellExecute 309
SendShellUTF8 309
SendSMAPIAppendLogs 314
SendSMAPIMessage 314
SendSMAPISubject 314
SendSMAPITarget 314
SendSMTPClientAppendLogs 316
SendSMTPClientFrom 316
SendSMTPClientHost 316
SendSMTPClientLogin 316
SendSMTPClientMessage 316
SendSMTPClientPassword 316
SendSMTPClientPort 316
SendSMTPClientSSL 316
SendSMTPClientSubject 316
SendSMTPClientTarget 316
SendSMTPClientTLS 316
SendSMTPClientUseRealEMail 316
SendSMTPServerAppendLogs 319
SendSMTPServerFrom 319
SendSMTPServerMessage 319
SendSMTPServerSubject 319
SendSMTPServerTarget 319
SendSMTPServerUseRealEMail 319
set debug paths tool 617
settings editor tool 617
shell 309, 391
ShellExecute 391
simple MAPI 314, 393
Smart Packer Pro 520
SMTP client 316, 397
SMTP server 319, 398
sndAddBugIDInFileName 304
sndAddComputerNameInFileName 304
sndAddDateInFileName 304
sndDeleteFileAfterSend 304
sndPack 304
sndScreenshot 304
sndSendEntireLog 304
sndSendInSeparatedThread 304
sndSendLastHTMLPage 304
sndSendXMLLogCopy 304
sndShowFailureMsg 304
sndShowSendDialog 304
sndShowSuccessBugClosedOnlyMsg 304
sndShowSuccessMsg 304
software exception 40

ssActiveWindow 304
ssApplication 304
ssDesktop 304
ssForegroundWindow 304
ssNone 304
ssPrimary 304
ssWorkarea 304
stack tracing 96
standalone settings editor tool 617
support 16
SupportURL 279
suspend 246

- T -

TD32 411
tech-support 16
TerminateBtnOperation 271, 279
Themida 520
threads 246
threads snapshot tool 617
tools 514, 515, 516, 519, 520, 617
troubleshotting 72, 596, 606, 614, 615
TThread 246
tutorial 20, 33, 38, 40, 45, 46, 52, 53, 55, 58, 68, 72, 79, 81, 105, 108, 115, 119, 134, 143, 180

- U -

unhandled exception 40
unit 40
update 4, 624, 627, 633, 636, 637
upgrade 4, 624, 627, 633, 636, 637
UPX 520
UseExceptionComments 244

- V -

viewer tool 617
visual studio 496, 527
VMProtect 520

- W -

web send 398, 404, 406, 407, 408
web tracker send 404, 406, 407, 408
WebErrorCode 296
welcome 2, 8
WERCustomizeReport 300
WERPassThroughUnexpected 300
WERPassThroughUnhandled 300
WERSubmitAddRegisteredData 300
WERSubmitHonorRecovery 300
WERSubmitHonorRestart 300
WERSubmitNoArchive 300
WERSubmitNoCloseUI 300
WERSubmitNoQueue 300
WERSubmitOutOfProcess 300
WERSubmitQueue 300
WERSubmitShowDebug 300
WERSubmitStartMinimized 300
what is EurekaLog 2, 8
what's new 4, 624, 627, 633, 636, 637
what's new in 4.5.x 637
what's new in 4.x 637
what's new in 5.x 636
what's new in 6.x 633
what's new in 7.x 4, 624, 627
Win32 96
Win64 96
WinLicense 520
wsmBugZilla 331
wsmFogBugz 322
wsmFTP 321
wsmHTTP 320
wsmJIRA 335
wsmMantis 327

- X -

x86-32 96
x86-64 96
XBundler 520
XComp 520
XPack 520

- Z -

ZipPassword 304