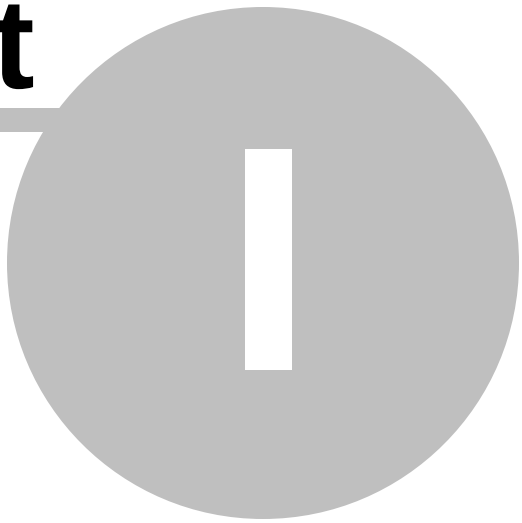
EurekaLog 6.x.x user manual

Table of Contents

Part



1 Introduction

1.1 What is EurekaLog

EurekaLog is the new add-in tool that gives your application (GUI, Console, Web, etc.) the ability to catch all exceptions and memory leaks, and generate a detailed log of the call stack with unit, class, method and line-number information as shown in the image below. The information shown is also logged to a disk file and may optionally be forwarded to you by e-mail or via Web (HTTP/S - FTP).

EurekaLog helps you find infinite loops and deadlock bugs, raising an exception when the application is frozen for a specified time. The exception shows the same application state as it does for other exceptions.
EurekaLog is easy to use because it is fully integrated into the Delphi/C++Builder IDE. You just rebuild your application to add this new exception-logging capability.

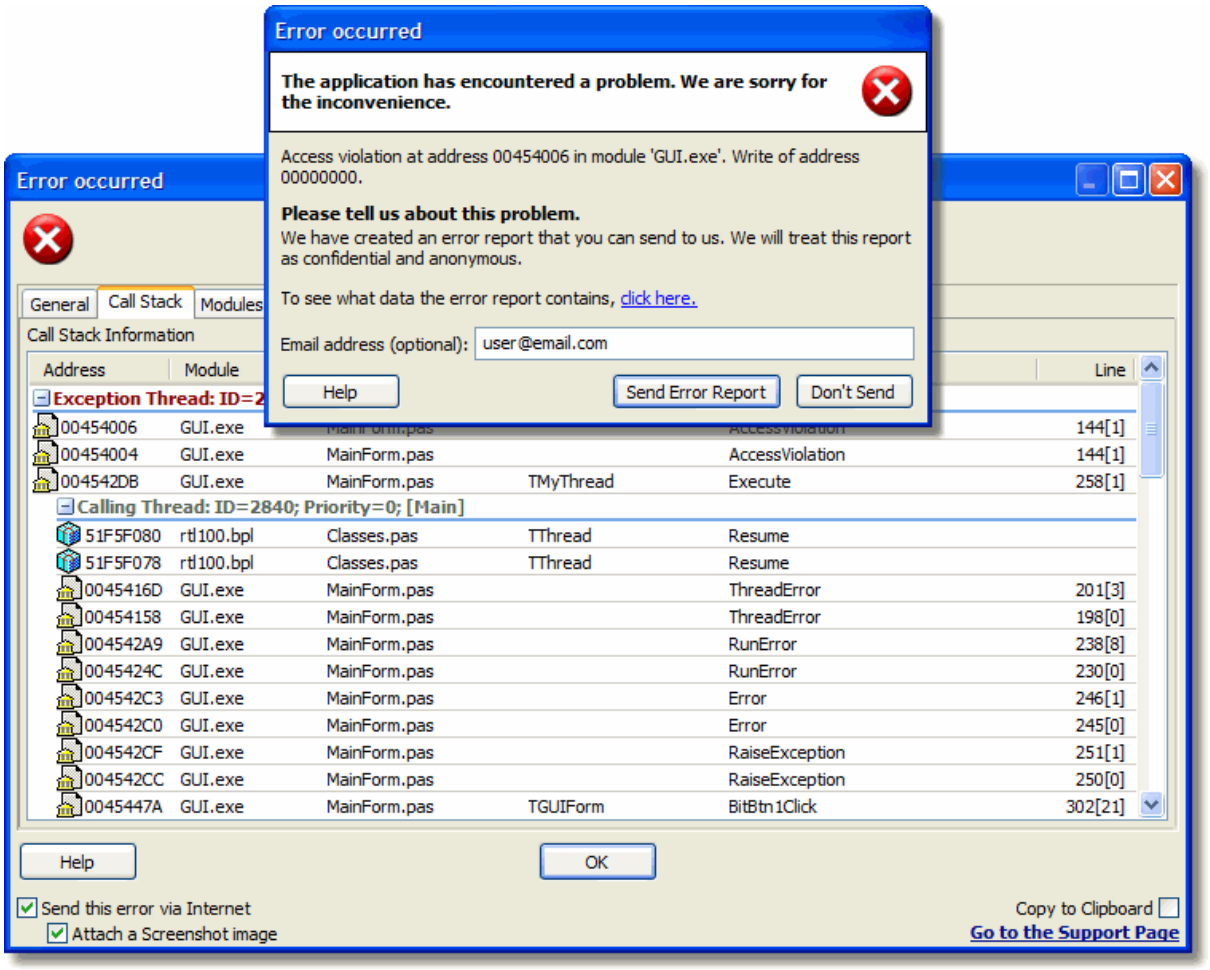
EurekaLog does not affect the performance of your application, as it only executes when an exception is raised. It increases the compiled file size by about 0.5% - 4% (this space is needed to store some additional, compressed and encoded, debugging information). To works EurekaLog needs only of the compiled file (not .map file).

EurekaLog is compatible with Delphi 3, 4, 5, 6, 7, 2005, 2006, 2007 and 2010, and with C++Builder 5, 6, 2006, 2007 and 2010. It works on all Windows platforms, from Win 95 to Win Vista.

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" 4 topic for further details.



A typical example of EurekaLog dialog.

1.2 Installation

Before installing EurekaLog, you must close all Delphi/C++Builder instances.

Run the EurekaLog installation program and select one or more Delphi/C++Builder versions you wish to install.

Start the installation.

When installation ends, open the Delphi/C++Builder IDE and you are ready to work with EurekaLog.

Note: EurekaLog install the TEurekaLog 64 component into the EurekaLog component palette.

1.3 How to use EurekaLog

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

EurekaLog will automatically apply all necessary changes to make your projects intercept every exception.

To access to your project EurekaLog options, use the new "Project/EurekaLog Options..." menu item (see here 11).

To check if your project is compiled with EurekaLog you can check if the *EUREKALOG* directive exists.

Example:

```
{$IFDEF EUREKALOG}
   WriteLn('Compiled with EurekaLog.');
{$ELSE}
   WriteLn('Compiled without EurekaLog.');
{$ENDIF}
```

To check if your project is compiled with a specified EurekaLog version you can check if the *EUREKALOG_VER5 or EUREKALOG_VER6* directive exists.

Example:

```
{$IFDEF EUREKALOG}
   {$IFDEF EUREKALOG_VER6}
     WriteLn('Compiled with EurekaLog 6.x.');
   {$ELSE}
     {$IFDEF EUREKALOG_VER5}
       WriteLn('Compiled with EurekaLog 5.x.');
     {$ELSE}
       WriteLn('Compiled with EurekaLog 4.x.');
     {$ENDIF}
   {$ENDIF}
{$ELSE}
   WriteLn('Compiled without EurekaLog.');
{$ENDIF}
```

1.4 Features

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

Source code	Pro	Ent
Full included Source Code	✗	✓

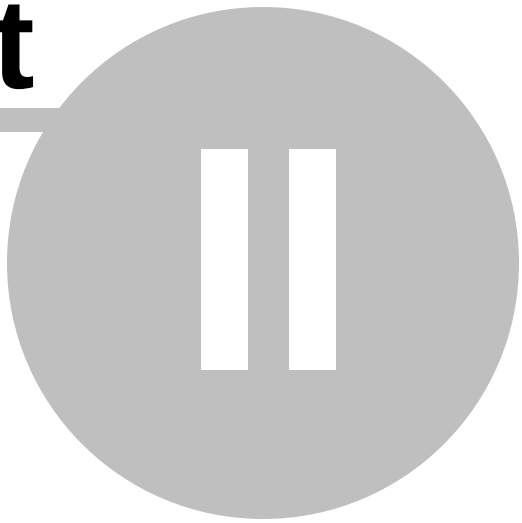
Common features	
Supported languages & Operating Systems	
Delphi versions	3-7 2005-2010
C++Builder versions	5-6 2006/2010
Windows 95/98/ME/NT/2000/XP/2003/Vista	✓
Compiled file & Debug data	
Kbytes added	~300
Overall increase	~0.5
Decrease in Application's performance without memory leaks detection	none

Decrease in Application's performance with memory leaks detection (% min - max)	0.5/5
128 bit debug data encryption *(to protect against crackers)*	✔
IDE & Compiler	
Full integration with Delphi/C++Builder IDE	✔
Help integrated in IDE	✔
Automatic search of error lines in modified sources *(guarantee error individuation in sources modified after the delivery)*	✔
New Design-Time exception management *(for debug packages at design-time)*	✔
Save on IDE crashes *(allows saving modified files after unrecoverable IDE crashes)*	✔
EurekaLog Viewer full integrated with the Delphi/C++Builder IDE	✔
The new EurekaLog Viewer is more similar to a stand alone BUG tracking tool ***NEW***	✔
Full revisited EurekaLog Options form ***NEW***	✔
New "IDE/Tools/EurekaLog IDE Settings" form ***NEW***	✔
Run custom programs before and after every project build ***NEW***	✔
History integration *(for the unit line searching on modified sources)* ***NEW***	✔
Options to reduce the .EXE file size ***NEW***	✔
Option to detect .EXE cracks ***NEW***	✔
Documentation	
CHM Help file	✔
Printable PDF manual	✔
On-line HTML manual	✔
New application capabilities	
Catches of every **EXCEPTION!!!**	✔
Catches of every **MEMORY-LEAKS!!!** (only on Delphi) ***NEW***	✔
Display the Assembler/CPU sections ***NEW***	✔
Display the Processes/Modules list sections ***NEW***	✔
Display of more Hardware and Software info (DPI, printer, VGA, privileges, ...) ***NEW***	✔
Full .jdbg (Jedi Debug file), .MAP and .TDS (TD32 Debug file) support ***NEW***	✔
Full customizables Exceptions Filters *(can choose style, behavior, messages ...)* ***NEW***	✔
Full COM and SafeCall Exceptions customization ***NEW***	✔
Optionally catches every **HANDLED EXCEPTION!!!** ***NEW***	✔
Environment variables (%EnvironmentVariable%) support ***NEW***	✔
Creation of a detailed Log for every exception *(module, unit, class, method, line ...)*	✔
Termination/restarting after a customizable number of exceptions	✔
Fully customizable exception management with new EurekaLog events	✔
Debug of third-party DLLs *(showing address, module and procedure)*	✔
Debug of third-party BPLs *(showing address, module, unit, class and procedure)*	✔
Initialization/Finalization exceptions are trapped *(only Delphi version)*	✔

Creation of a detailed log when the application freezes *(as for any other exception)*	✔
Full Call-Stack customization	✔
Display detailed running Threads Call-Stack *(calling Thread Call-Stack included)*	✔
Exception dialogs & HTML error page	
Full customizable Exception-Dialog *(with more new styles - as MS style)* ***NEW***	✔
Add a customizable HELP button *(call an event)* ***NEW***	✔
Full UNICODE logs handling ***NEW***	✔
Fully customizable message texts collections *(for multi-language applications)* ***NEW***	✔
Adding customizable "Support Link" in the Exception-Dialog	✔
Customizable HTML error page via HTML template	✔
Automatic Exception-Dialog closing after a customizable time	✔
Jump from Exception Dialog line to source code line *(with a simple double-click)*	✔
Customizable log-file section view *(show/hide Modules-Processes & Assembler-CPU sections)*	✔
Advanced tree view of all running/calling threads call-stacks	✔
Send messages & Files	
Delivery of every new BUG to the most used Web BUG-Tracking tools ***NEW***	✔
Compress and encrypt all files to send in in ZIP format ***NEW***	✔
Delivery of a customizable email or Web message for every exception, with log-file	✔
Attach a PNG Screenshot to the message	✔
Append text containing a user description on the bug reproducibility to the log	✔
Upload of log-file and attached files via HTTP/HTTPS and FTP protocol	✔
Send a log-file copy in XML format	✔
Send the last generated HTML page *(only for the WEB applications)*	✔
Send the email/"upload files" in a separated thread	✔
Attach customizable files to the send message	✔
Add customizables data to the log-file	✔
Add customizables fields to the uploading HTML page	✔
Type of supported applications	
MultiThread	✔
Indy threads/components	✔
GUI *(standard graphical applications)*	✔
Console	✔
BPL *(Borland package library)*	✔
DLL *(dynamic-link library)*	✔
NT Services	✔
Control Panel	✔

ActiveX-COM	✔
ISAPI	✔
CGI	✔
IntraWeb	✔
Applications compiled with Runtime Packages	✔
Other	
Delphi Command-line compiler	✔
C++Builder Command-line compiler	✔
FinalBuilder support	✔
Exe compressors full compatibility	✔
Exe protectors full compatibility	✔
New EurekaLog Viewer full integrated with the Windows Shell	✔

Part



2 What's New

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" 97 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!

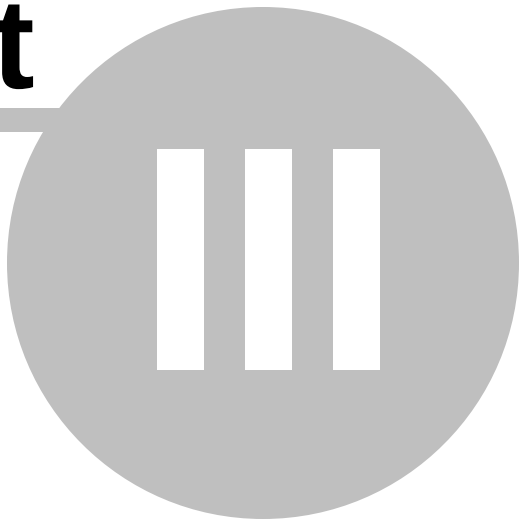
Compatibility issues:

- For any "Compatibility Issues" try to see the "Changed from the old 5.x version" 85 section!

Features list:

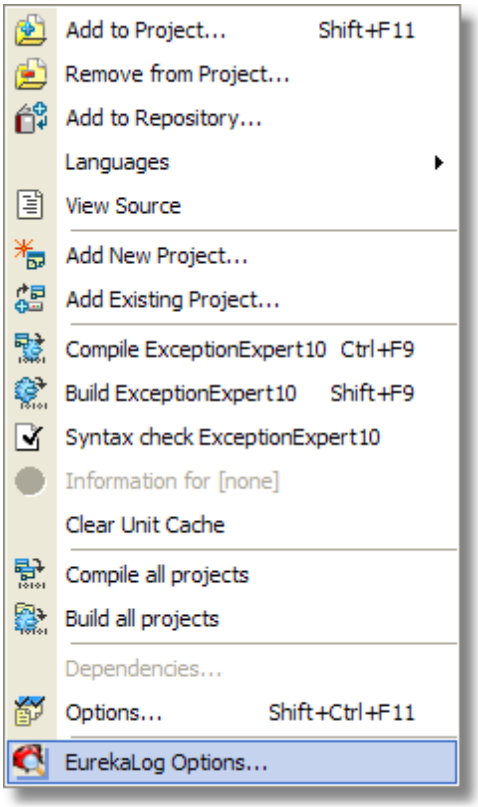
- For a more detailed features list, see the "Features" 4 page.

Part

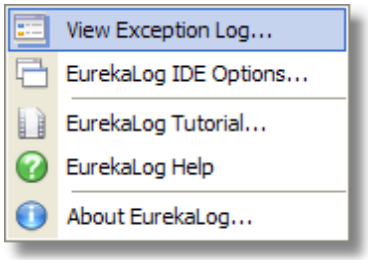


3 New options

3.1 New menu options

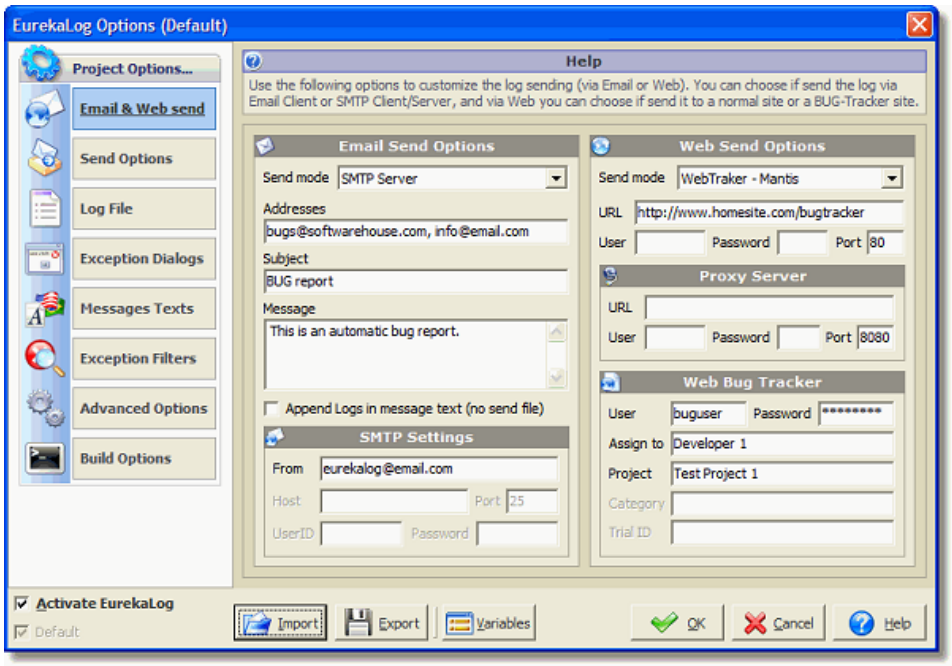


New options in IDE's Project menu, added by EurekaLog: click on "EurekaLog Options..." to access EurekaLog options.



New options in IDE's main menu added by EurekaLog to can use all its functions.

3.2 Email & Web Send Tab



Here you may tell the program to automatically send an email (multiple addresses are allowed) a Web message or a bug report to a Web BUG-Tracking tool (like BugZilla, Mantis or FogBugz) whenever an exception is raised.

Email Send Options:

You can choose to send the email via *default email client*, via *SMTP client* or via *internal SMTP server*.

If you select the *SMTP client* option then you must provide the relevant settings (email from address, host name/ip, host port and UserID/Password, the later only if the SMTP server requires log-in).

If you select the *SMTP server* option then you must provide the relevant settings (email from address).

For all *Email* options, you may request to *"Append Logs in message text"* adding the Log text at the bottom of the email message.

Web Send Options (to a generic web site):

Activating this options you may tell to the program to send the Log-File with all the other attached files to a Web location.

You can choose to send the message via *HTTP*, *HTTPS* (secure HTTP) or *FTP* Internet protocol.

You must set a valid URL and optionally the User and Password fields.

To configure a custom Proxy setting you can use the "Proxy Server" box.

Follow a PHP Web server script capable to download all the HTTP/HTTPS uploaded files via

the Web send feature.

PHP example:

```
<?
  foreach ($_FILES as $key=>$value)
  {
    $uploaded_file = $_FILES[$key]['tmp_name'];
    $server_dir = 'upload/';  // Upload folder
    $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 your code...
    }
  }
?>
```

Web Send Options (to a BUG-Tracking tool):

Activating this options you may tell to the program to send the Log-File with all the other attached files to a Web BUG-Tracking tool (like BugZilla, Mantis or FogBugz).

You can send the bug report via *HTTP* or *HTTPS* (secure HTTP).

To set all the Web BUG-Tracking parameter use the "Web BUG Tracker" box.

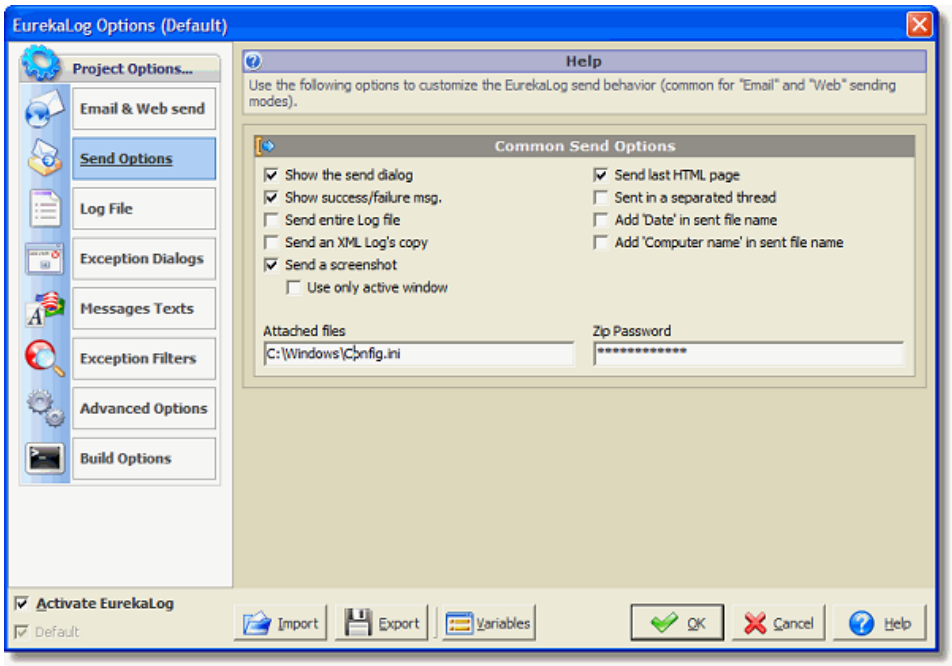
New Events:

To send to a Web HTML page a specified fields list, it's available the new CustomFieldsRequest 43 event.

To add some custom data to the Log text it's available the new CustomDataRequest 42 event.

To send custom attached files with the message (Email and Web) it's available the new AttachedFilesRequest 41 event.

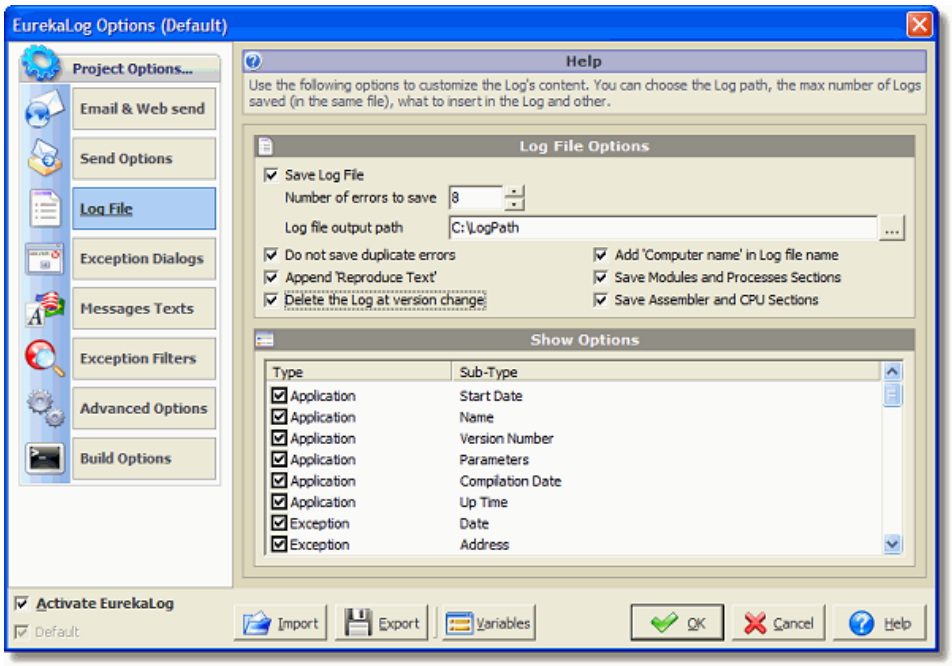
3.3 Send Options Tab



Common Send Options:

"Show the send dialog"	Show the send dialog
"Show success/failure msg."	Show the success or failure message after the send process
"Send entire Log file"	Send the entire log *(by default only the log of the latest exception is sent)*
"Send an XML Log's copy"	Send an XML copy of the Log file
"Send a screenshot"	Attach a PNG screenshot to the message
"Use only active window"	Catches a screenshot of active window instead that the entire desktop
"Send last HTML page"	Send the last generated HTML page *(for Web application only)*
"Show in a separated thread"	Run the send process in a separated thread *(so the main process can continue without any other break)*
"Add 'Date' in sent file name"	Add the current date-time *(in ISO format)* to the sent file name
"Add 'Computer name' in sent file name"	Add the Computer name to the sent file name
"Attached files"	A list of files path to attach to the sending message
"Zip Password"	An optionally password used to encrypt the sent ZIP file

3.4 Log File Tab



In this section you may customize the log file.

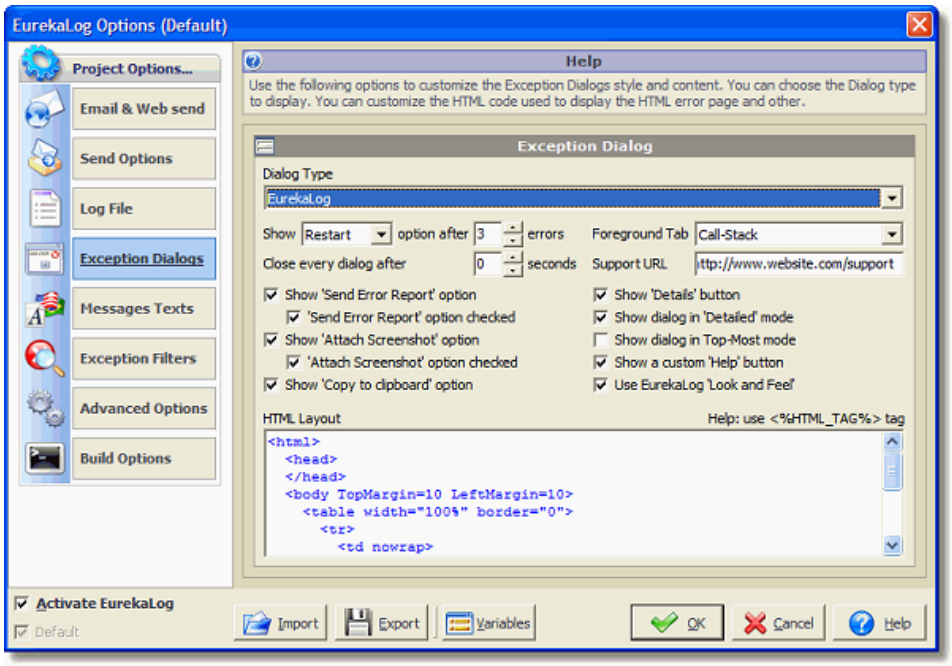
Saving Options:

"Save Log File"	Save the log file into the "Output path" folder
"Number of errors to save"	Max number of logged exceptions to save (min=1)
"Output Path"	The full path of Log File (if is empty then the log is saved into the current program path - if the folder is not writable then the log is saved in the "%UserProfile%/EurekaLog/ProjectName" folder)
"Do not save duplicate errors"	Inhibit logging exceptions that have already been logged to the log file *(Inhibiting it's send too)*
"Append 'Reproduce Text'"	Append to the Log File a text, entered by the user of your program, describing How to reproduce the error
"Delete the Log at version change"	Delete the Log file when the program version it's changed
"Add 'Computer name' in Log file..."	Add the Computer-Name to the Log file name
"Save Modules and Processes Sections"	Save the Modules and Processes sections in the Log File *(displaying it in the Exception Dialog)*
"Save Assembler and CPU Sections"	Save the Assembler and CPU sections in the Log File *(displaying it in the Exception Dialog)*

Show:

You may select the type (Application, Exception, etc.) and sub-type (Name, Date, Module, etc.) of the exceptions you wish to store in the log-file.

3.5 Exception Dialogs Tab



In this section you may customize the Exception Dialog (or HTML error page for the Web application).

Dialog Type:

You can choose the Exception Dialog from:

- none
- MessageBox
- MS Classis
- EurekaLog Dialog

Terminate/Restart option:

You can choose to Terminate or Restart the application after a specified number of exceptions.

Foreground option:

You can indicate which Tab, General, Call-Stack, Modules, Processes, Assembler or CPU, should be presented in the foreground when an exception is presented on screen.

Close Dialog option:

You can choose to close every Exception Dialog after a specified number of seconds that the dialog is unused.

Support URL option:

You can set this option to display in the Exception Dialog a specified Internet link (URL).

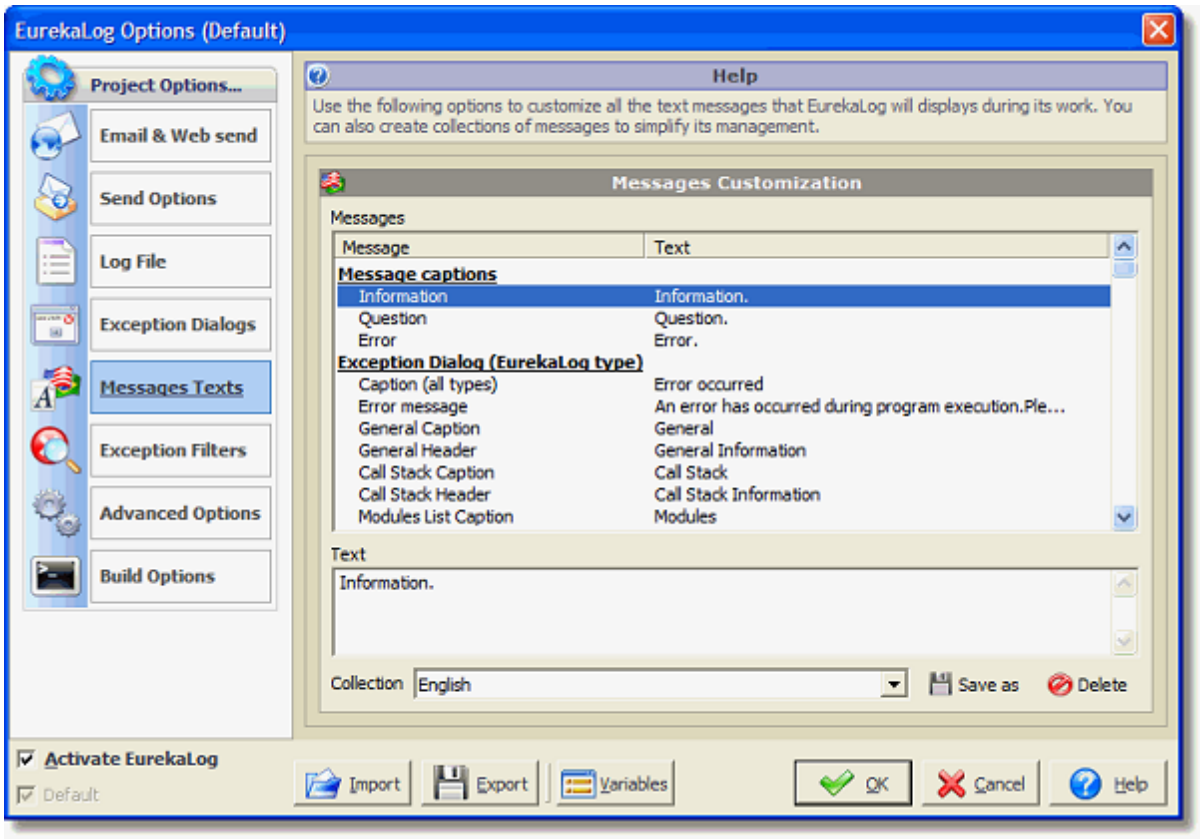
Common Options:

"Show 'Send Error Report' option"	Show the 'Send Error Report' option
"'Send Error Report' option checked"	Check the 'Send Error Report' option
"Show 'Attach Screenshot' option"	Show the 'Attach Screenshot' option
"'Attach Screenshot' option checked"	Check the 'Attach Screenshot' option
"Show 'Copy to clipboard' option"	Show the 'Copy to clipboard' option
"Show 'Details' button"	Show the 'Detail' button *(used to diaplay all the detailed exception data)*
"Show dialog in 'Detailed' mode"	Show the Exception-Dialog as the customer has pressed the 'Detail' button
"Show dialog in Top-Most mode"	Show the Exception-Dialog in Top-Most mode *(in Top to all displayed windows)*
"Show a custom 'Help' button	Show a customizable 'Help' button in the main Exception Dialog (you can customize its text via "Messages text Tab 18" and its behavior via the "CustomButtonClickNotify 45" event)
"Use EurekaLog 'Look and Feel'"	Display the Exception-Dialog using it's personal 'Look and Feel' *(ignoring the standard Windows 'Look and Feel')*

HTML Layout:

A custom HTML code used to show the HTML error page (Web application only).
Use the <%HTML_TAG%> to indicate where the error message will be show.

3.6 Messages Texts Tab



Here you can customize the texts of EurekaLog so you can use EurekaLog in your favorite language, or just to show a customized text.

First select a message in the "Messages" window and then customize it in the "Text" box underneath.

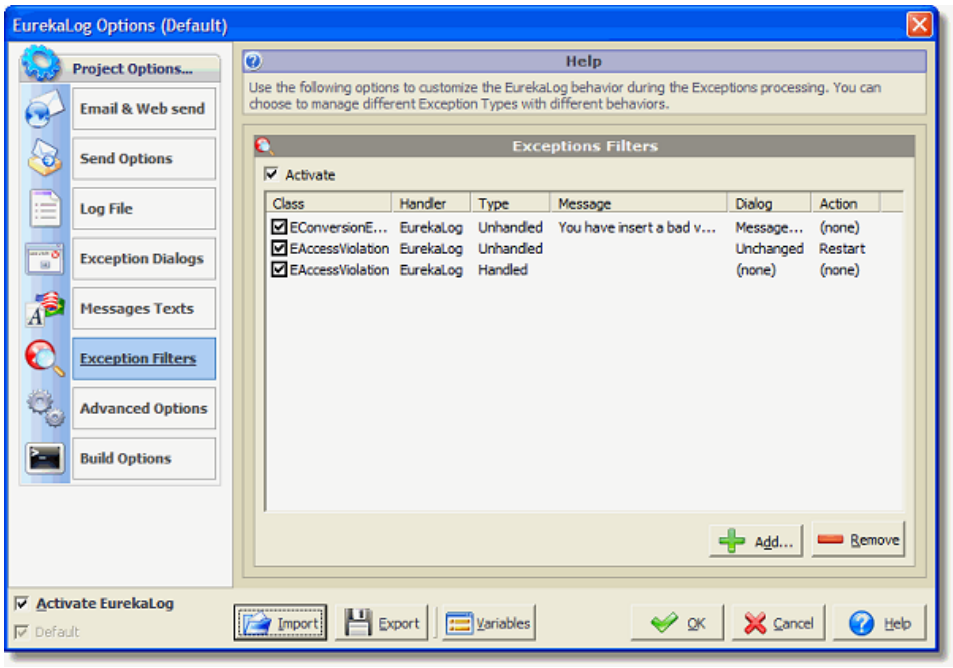
You can save every your "Messages Text Collection" using the "Save as" button.

You can delete a "Messages Text Collection" using the "Delete" button.

You can load a saved "Message Text Collection" simply selecting it from the drop-down list.

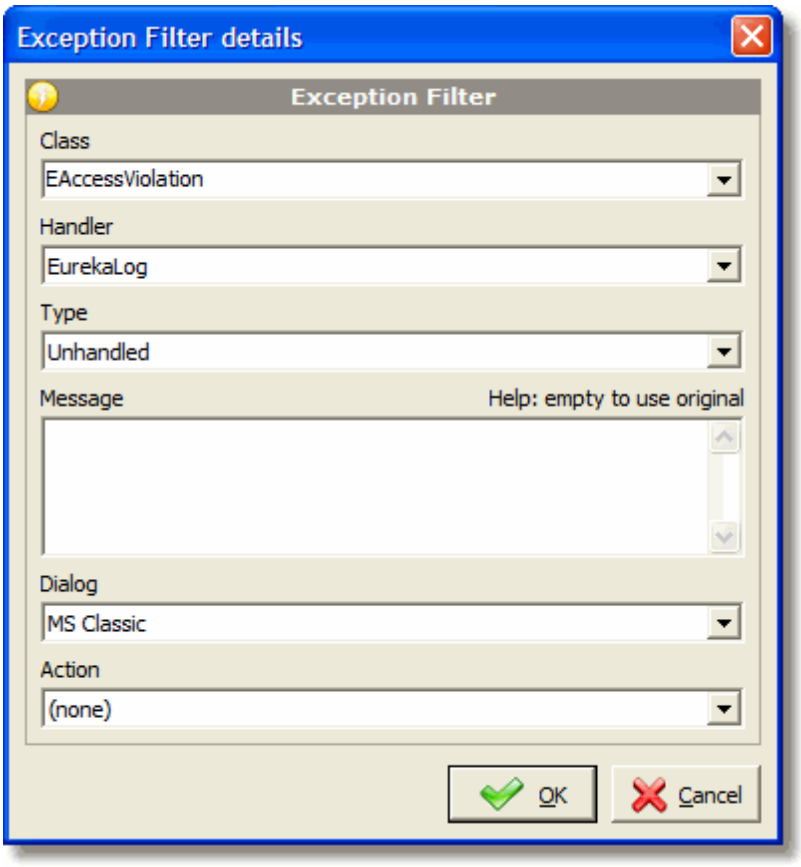
Using the previous options you can also create and save more different EurekaLog translations.

3.7 Exception Filters Tab



With the options on this tab you can full customize the exception types that should be processed in a different way.

When an exception on this list is intercepted, it will not be processed in a normal way but following the specified Exception Filter setting.



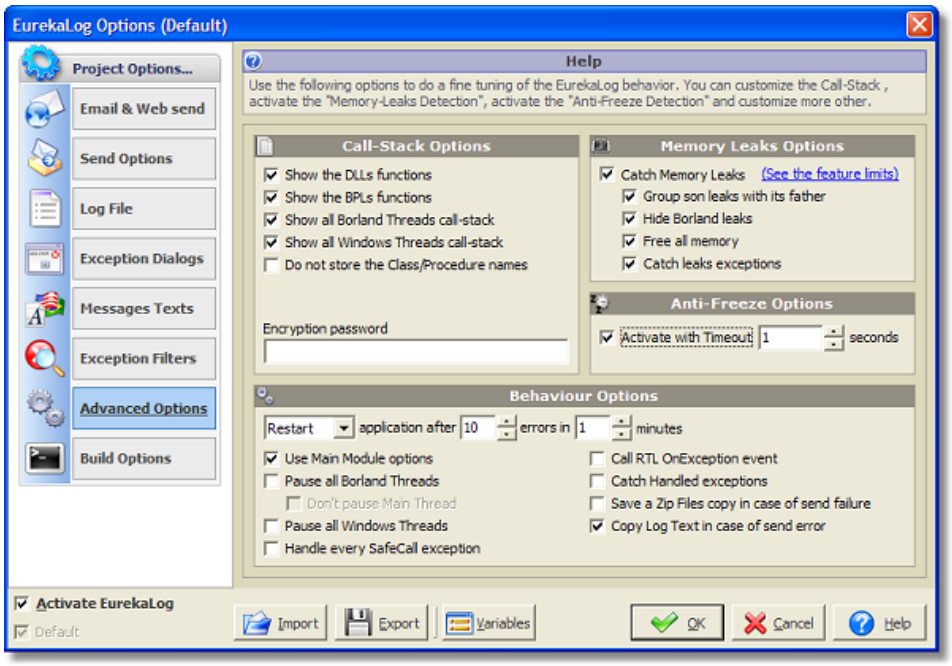
Is possible choose:

- the Filter's Exception Type
- the Exception Handler (none = the exception will be ignored, RTL = the exception will be processed by Delphi/C++Builder; EurekaLog = the exception will be processed by EurekaLog)
- The Type of Exception to process (Unhandled, Handled);
- The Message to display (empty to use the original Exception message)
- the Exception Dialog to use (none, Unchanged, MessageBox, MS Classic, EurekaLog)
- the Action to execute (none, Terminate, Restart).

Use the "Add" key to add a new Exception Filter. Use the "Remove" key to remove the selected Exception Filter.

Use the "Activate" option to activate/deactivate this feature.

3.8 Advanced Tab



In this section you may customize all the available "Advanced Options".

Call-Stack Options:

"Show the DLLs functions"	Display all the DLLs functions contained in call-stack
"Show the BPLs functions"	Display all the BPLs functions contained in call-stack
"Show all Borland Threads call..."	Display all the running Borland Threads functions contained in call-stack
"Show all Windows Threads call..."	Display all the running Windows Threads functions contained in call-stack
"Do not store the Class/Procedure..."	Do not store in the Debug data the Class/Procedure names. This will increase the security and will reduce a bit the compiled file size *(reducing the Call-Stack readability)*
"Encryption Password"	Password use to encrypt all the debug data included in the compiled file *(the password isn't stored in the final compiled file so never hacker can use the debug information to crack your software).* You can use the "EurekaLog Viewer" 82 to decrypt the encrypted Exception Logs.

NOTE: See the "PasswordRequest" 44 event for further details about the encryption method.

Memory-Leaks Options:

In this panel, you can activate the EurekaLog Memory-Leaks detection and all its related options.

"Group son leaks with its father"	Group all the derived leaks with the leak that has generated them
"Hide Borland leaks"	Hide all the leaks derived from Borland code
"Free all memory"	Free all memory leaved unused by the leaks

"Catch leaks exceptions"	Catch every advanced leaks exceptions *(as double free and memory overrun)*

NOTE: see the "Memory Leaks Limit" 97 page for further details.

Anti-Freeze Options:

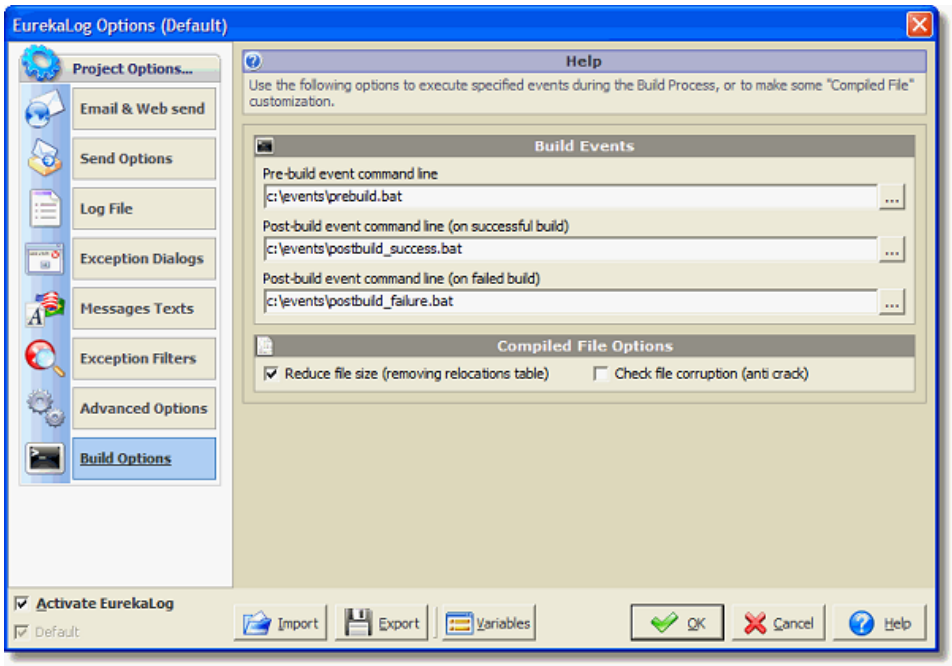
In this panel, you can select how EurekaLog detects that your application has frozen. You can choose to activate this feature, set the timeout needed to establish that the application is frozen (to customize the message text to be shown when the application freezes use the "Messages Text Tab" 18).

If EurekaLog detects a frozen application, it raises an "EFrozenApplication" 80 exception with your custom message text.

Behaviour Options:

"Auto Terminate/Restart"	Terminate/restart the application after a customizable crashed number
"Use Main Module options"	Use the EurekaLog options of Main Module *(only for the library modules)*
"Pause all Borland Threads"	Pause all Borland Threads during the Exception Dialog displaying
"Don't pause Main Thread"	Do not pause the Main Thread during the Exception Dialog displaying
"Pause all Windows Threads"	Pause all Windows Threads during the Exception Dialog displaying
"Handle every SafeCall exception"	Force the processing of every SafeCall exception
"Call RTL OnException event"	Call the RTL OnException event after than EurekaLog has processed every exception
"Catch Handled exceptions"	Catch every HANDLED exception (for this exceptions type EurekaLog did not call the "ExceptionNotify" 36 event but the new "HandledExceptionNotify" 37 event)
"Copy Log in case of wrong sending"	Copy the Log text into the clipboard in case of wrong sending
"Save a Zip Files copy in case of..."	Save a compressed ZIP copy of files to send in case of wrong sending

3.9 Build Options Tab



Build Event Options:

You can choose to run a program before and after ever project build.

Is possible choose a different program to execute:
- before a build process
- after a successful build process
- after a failed build process

Compiled File Options:

"Reduce the file size (removing ..."

Reduce the final compiled file size, removing the "relocations table" which reduced it by 4KB, or 7% *(valid only for non-libraries file)*

NOTE: this option may cause error if used with external localizer software.

"Check file corruption (anti crack)"

Check the file integrity at its startup raising a simulate exception in case of its crack (the exception message is customizable via "Messages Text Tab 18")

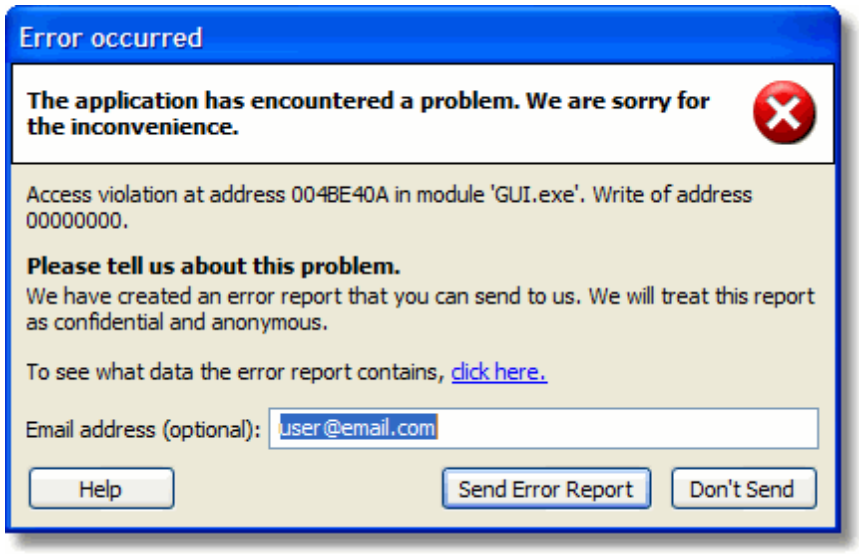
NOTE: this option may cause false-positive error if used with software that modified the final compiled file (as EXE compressors/protectors).

Part

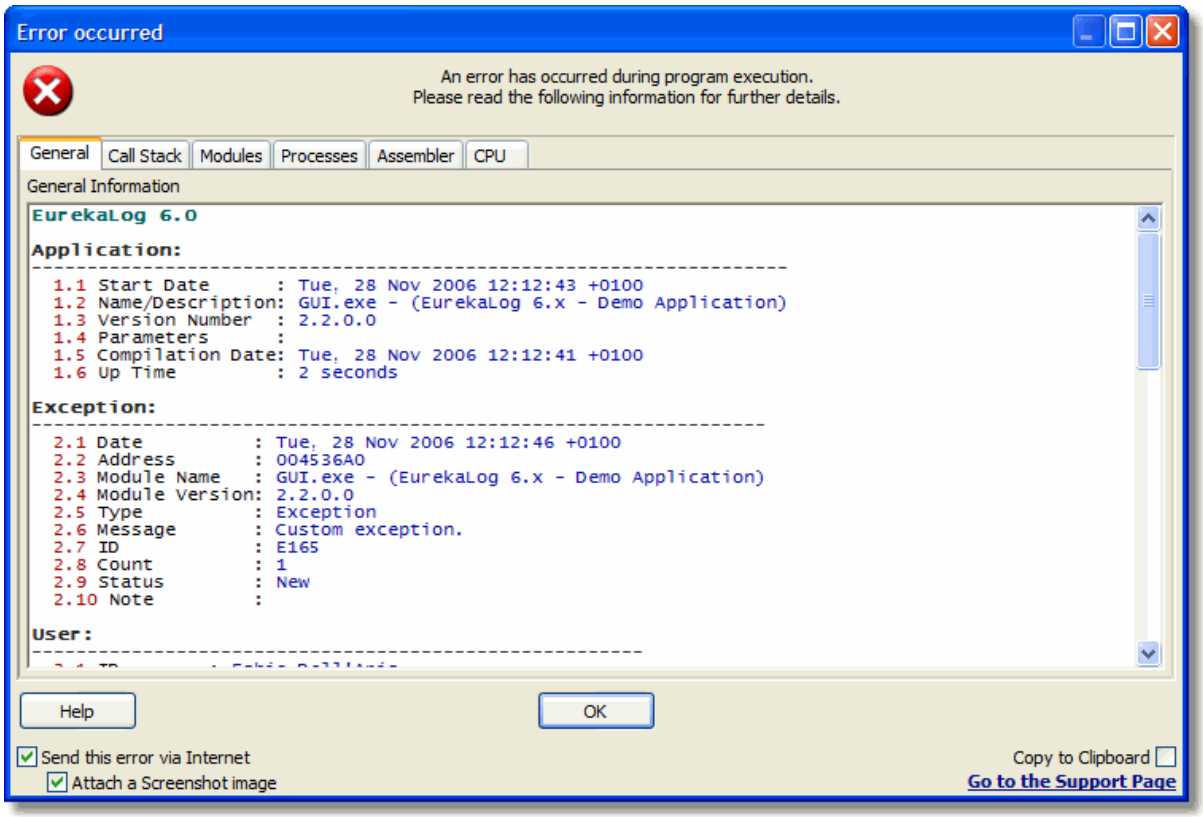
4 Types of Application

4.1 GUI

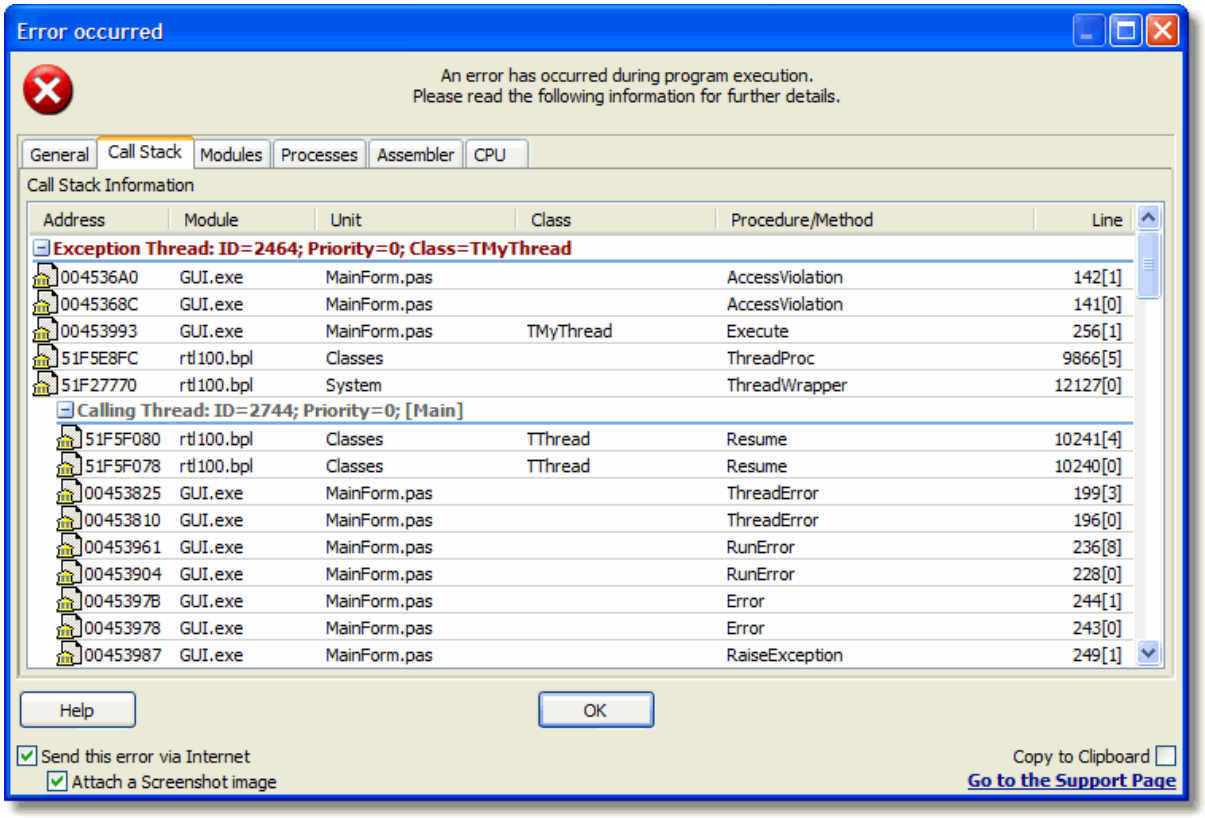
Information displayed when EurekaLog runs in a GUI (standard graphical) application:



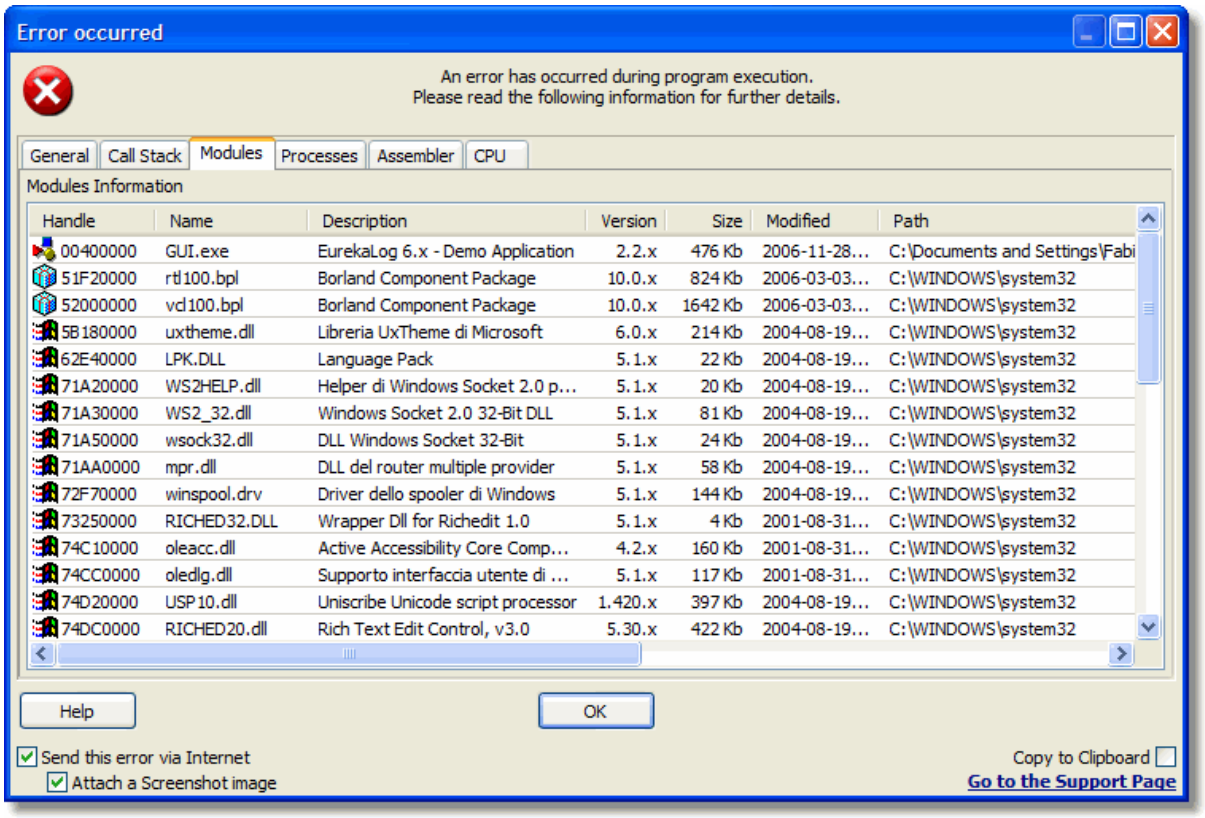
Pressing the "click here" link you can show detailed information about the exception.



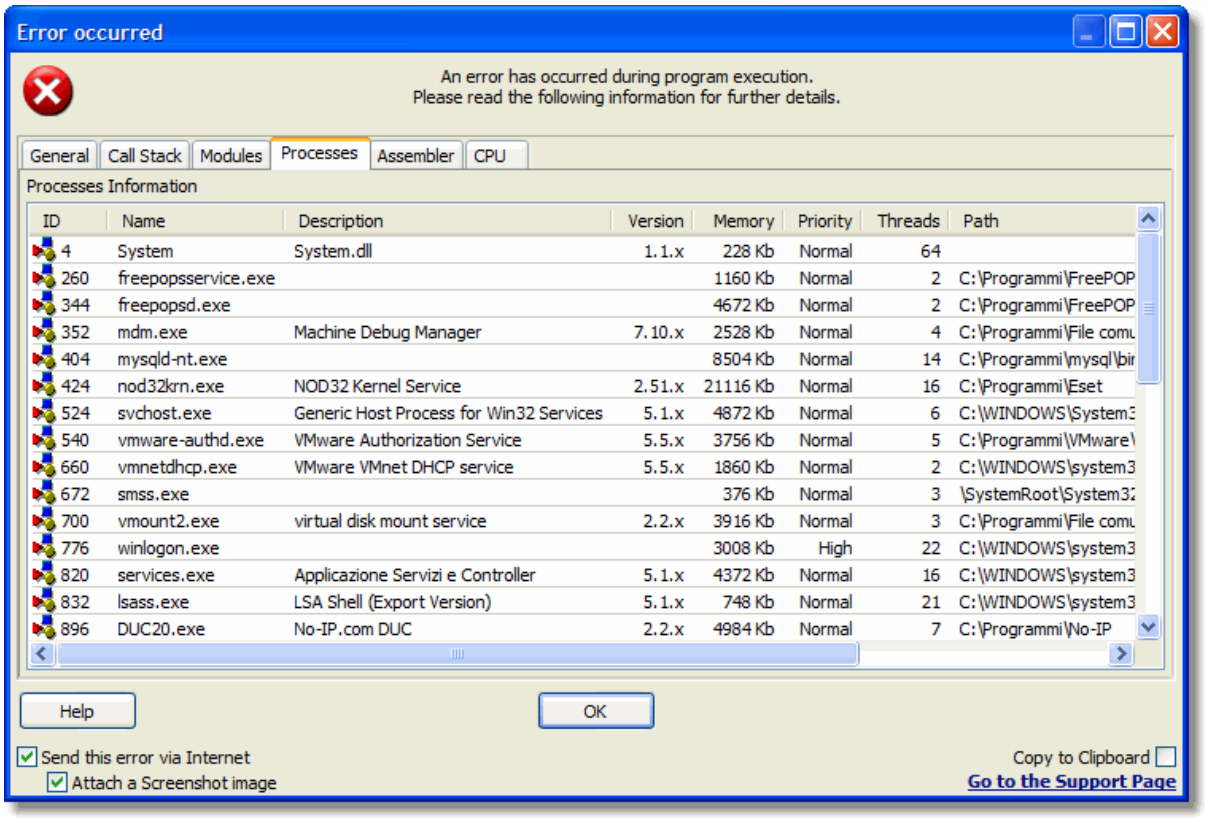
General information.



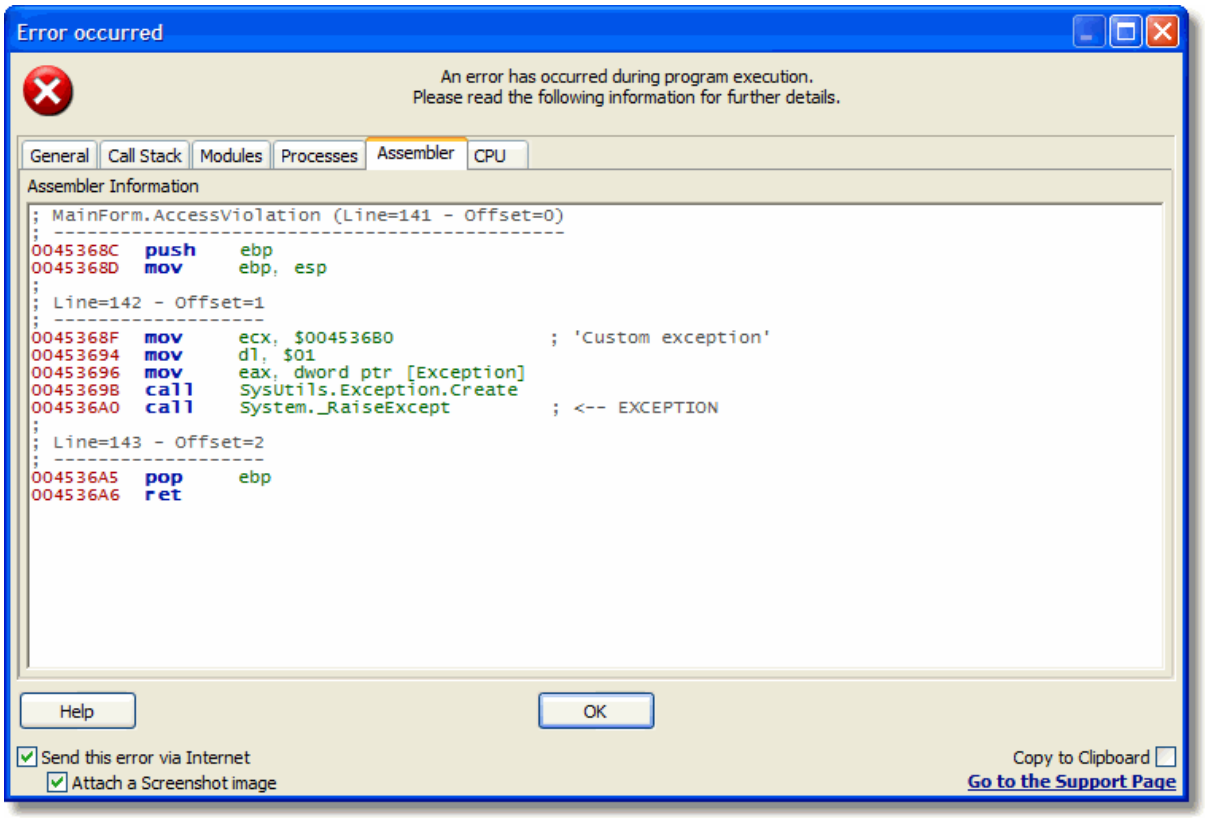
Call-Stack information for any running thread and relative calling thread.



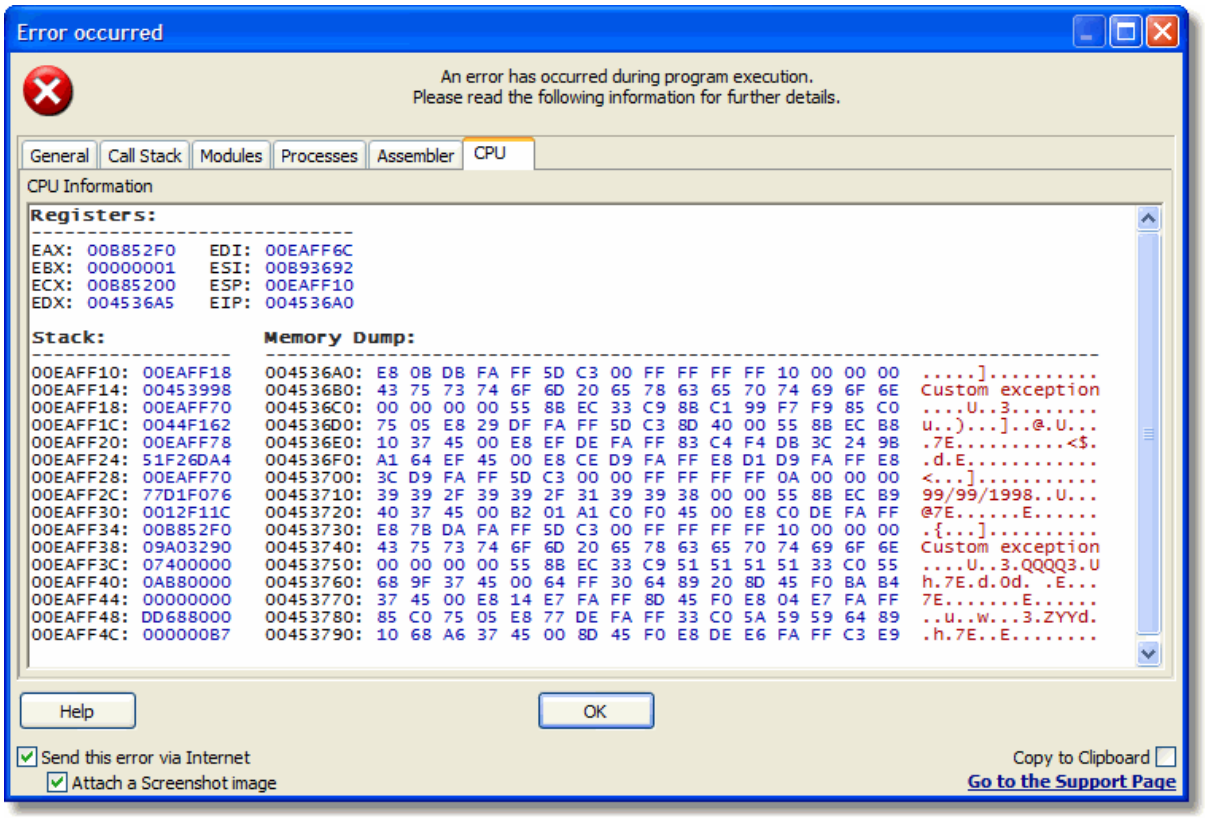
The Modules list.



The Processes list.



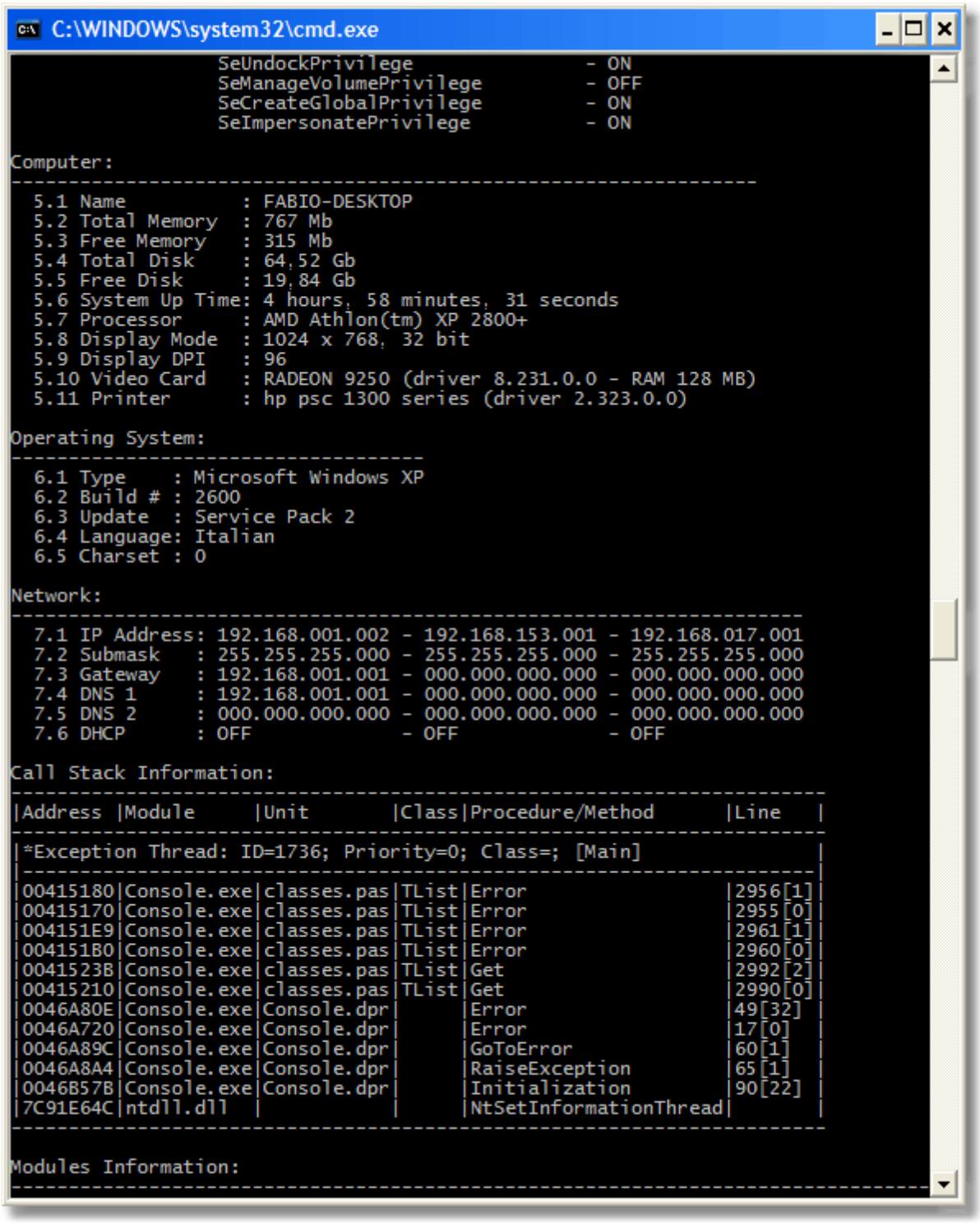
Dis-Assembler details.



CPU details.

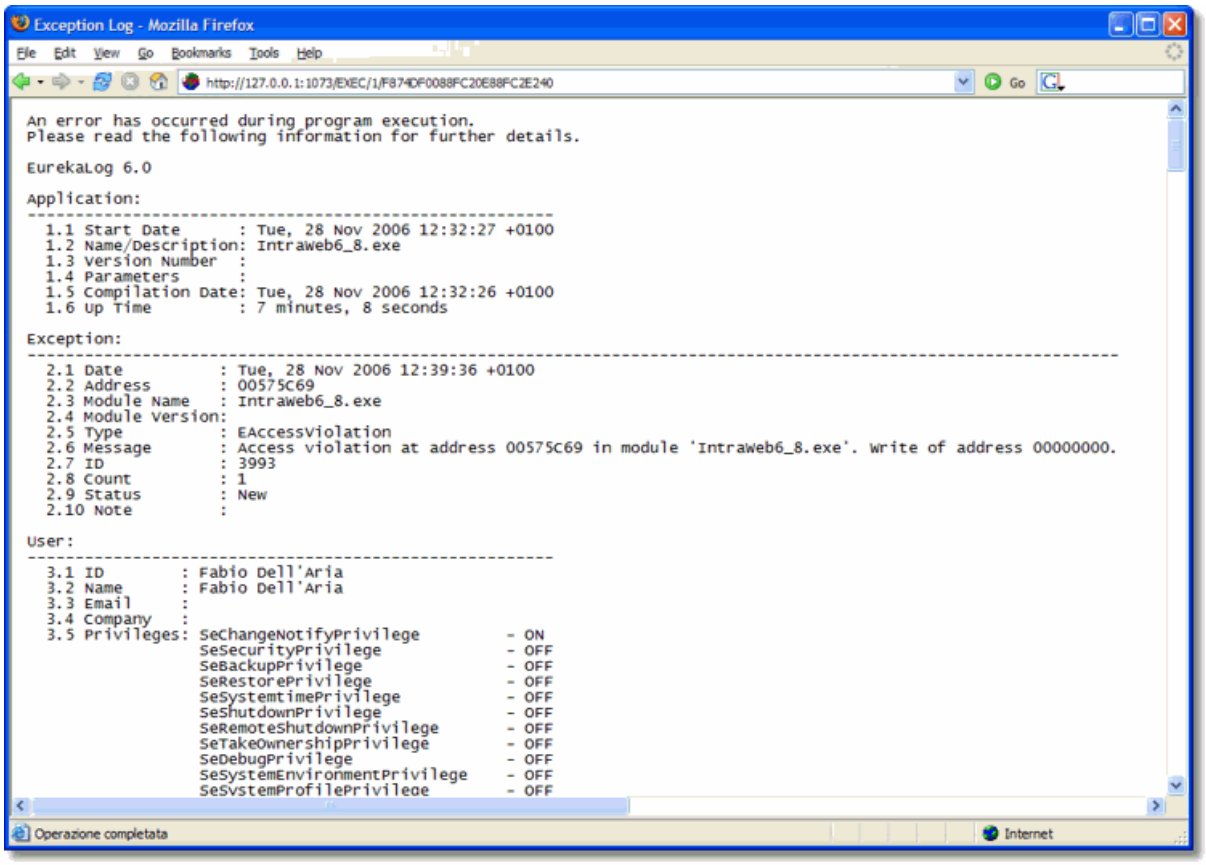
4.2 Console

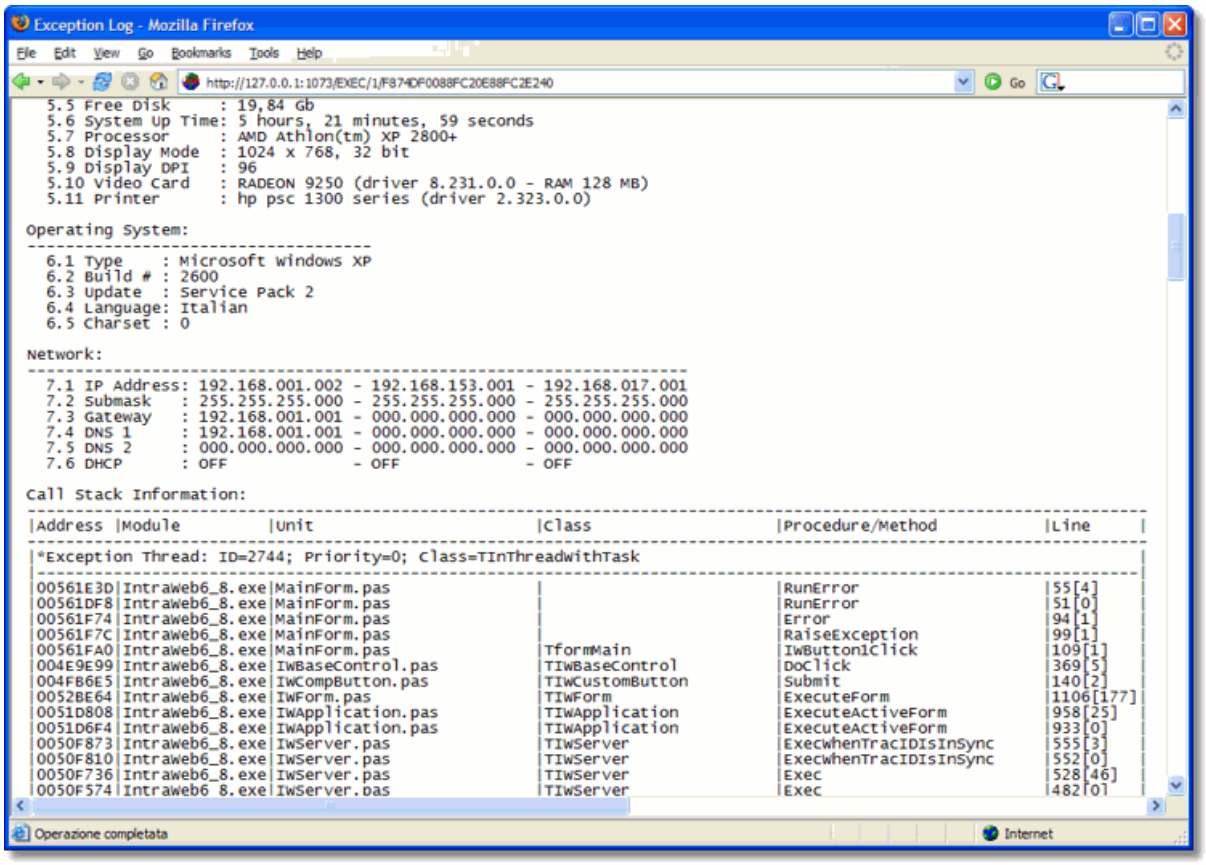
EurekaLog displays this information when running in a Console application:



4.3 Web

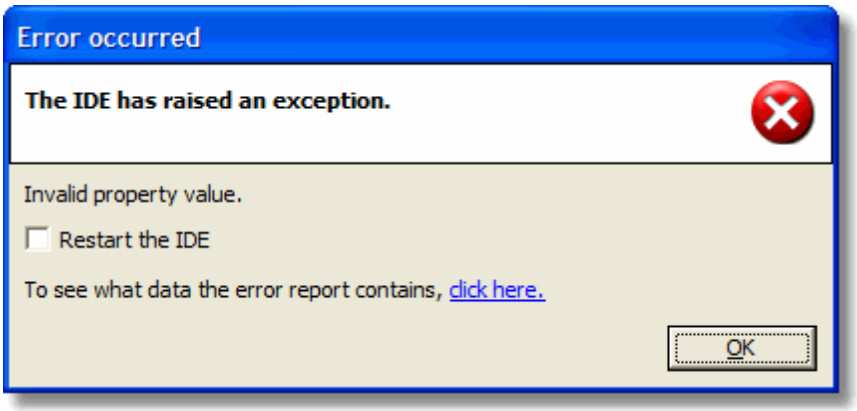
Information shown by EurekaLog when it runs in a Web application:



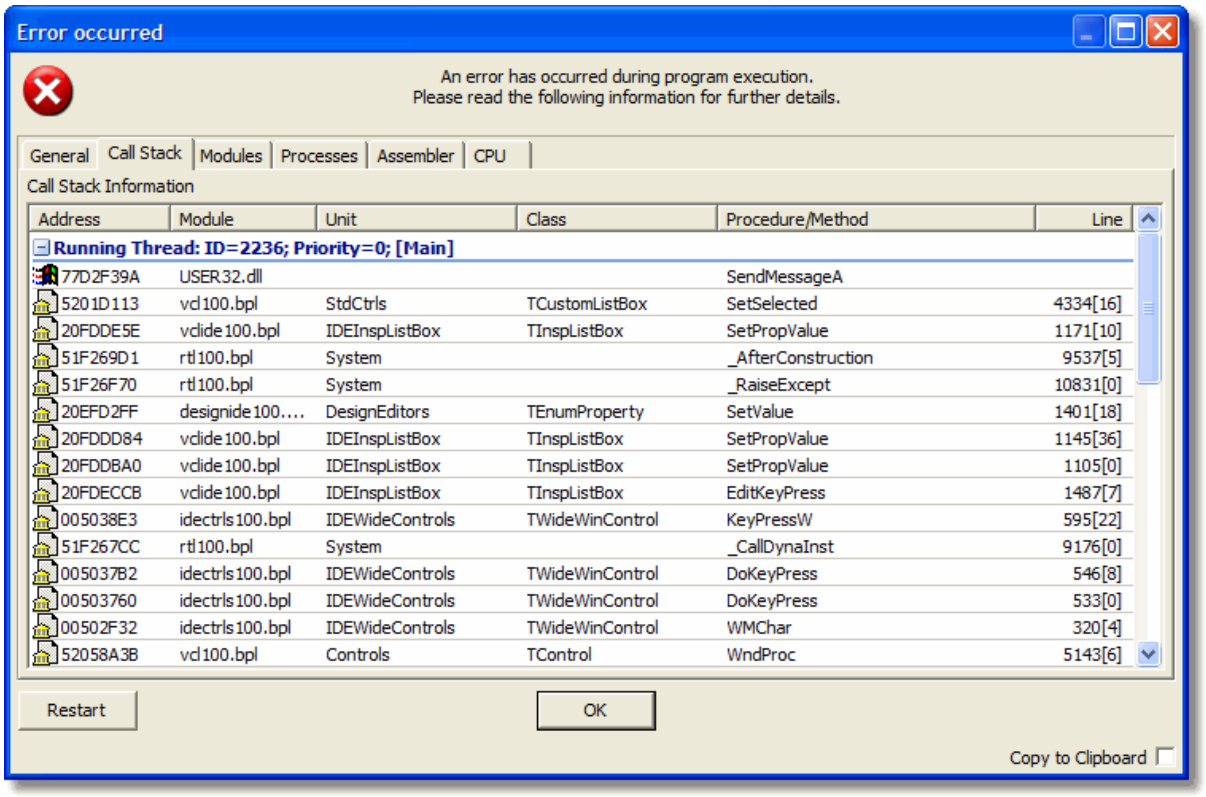


4.4 Design-Time

Dialog shown by EurekaLog when an exception is intercepted in the Delphi/C++Builder IDE:



Pressing the "Details" button shows detailed information about the exception.
Pressing the "Terminate" button, you can terminates your Delphi/C++Builder session saving all modified files (very usefully when the IDE craches).



This window shows all VCL and User Source Code points executed before the raised exception. You can debug Delphi/C++Builder BPLs with the IDE using this information.

Part V

5 Events

5.1 ExceptionNotify event

5.1.1 How to use this event

EurekaLog includes an "ExceptionNotify" event for interaction with the user program when an exception is raised. The user-defined routine is called when EurekaLog traps an exception.

The syntax of this event is :

```
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
```

To use this event, you must declare a procedure with those parameters and assign it to the ExceptionNotify EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   ...
   ...  // Your code...
   ...
end;

begin
   ExceptionNotify := MyNotify; // Assign ExceptionNotify variable to MyNotify procedure.
end.
```

If you set the *"Handled"* parameter to *False* in your routine before returning, the exception will be reported by the standard Borland Exception Manager. If you set *"Handled"* to *True* (the default value) the exception will be handled by EurekaLog.

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnExceptionNotify event.

5.1.2 Examples

Some examples:

```
// Check unit name...
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   // Check for SourceCode information...
   if (ExcRecord.CallStack[0]^.DebugDetail = ddSourceCode) and
     (ExcRecord.CallStack[0]^.UnitName = 'System.pas') then Handled := False;
end;


// Show exception dialog?
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   with ExcRecord.CurrentModuleOptions do
   begin
     if (MessageBox(0, 'Do you want to show error details?', 'Question',
                    MB_YESNO or MB_ICONQUESTION or MB_TASKMODAL) = ID_YES) then
```

```
        ExceptionDialogType := edtEurekaLog // Replace this with the option you wish to use for the
      else
        ExceptionDialogType := edtNone;
    end;
end;


// Don't save the Log File...
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   ExcRecord.CurrentModuleOptions.SaveLogFile := False;
end;


// Don't handle this exception with EurekaLog but with standard Borland Exception Manager...
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   Handled := False;
end;


// Check exception type...
procedure MyNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
var Error: Exception;
begin
  if ExcRecord.ExceptionObject is Exception then
     begin
       Error := Exception(ExcRecord.ExceptionObject); // Obtain exception type...
       MessageBox(0,PChar(Format('Error: %s',[Error.Message])),'Error',
                  MB_OK or MB_ICONSTOP or MB_TASKMODAL);
       ExcRecord.CurrentModuleOptions.ExceptionDialogType := edtNone; // Don't show exception dial
     end;
end;
```

5.2 HandledExceptionNotify event

5.2.1 How to use this event

EurekaLog includes a "HandledExceptionNotify" event for interaction with the user program when an HANDLED exception is raised (for the UNANDLED exceptions see the "ExceptionNotify 36" event). The user-defined routine is called when EurekaLog traps an HANDLED exception.

The syntax of this event is :

```
procedure MyHandledNotify(ExcRecord: TEurekaExceptionRecord 69; var Handled: Boolean);
```

To use this event, you must declare a procedure with those parameters and assign it to the HandledExceptionNotify EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyHandledNotify(ExcRecord: TEurekaExceptionRecord 69; var Handled: Boolean);
begin
   ...
   ...  // Your code...
   ...
end;

begin
```

```
   HandledExceptionNotify := MyHandledNotify; // Assign HandledExceptionNotify variable to MyHandl
end.
```

If you set the *"Handled"* parameter to *False* in your routine before returning, the exception will be reported by the standard Borland Exception Manager. If you set *"Handled"* to *True* (the default value) the exception will be handled by EurekaLog.

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnHandledExceptionNotify 64 event.

5.2.2 Examples

Some examples:

```
// Don't save the Log File...
procedure MyHandledNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   ExcRecord.CurrentModuleOptions.SaveLogFile := False;
end;


// Check exception type...
procedure MyHandledNotify(ExcRecord: TEurekaExceptionRecord; var Handled: Boolean);
var
   Error: Exception;
begin
   if ExcRecord.ExceptionObject is Exception then
     begin
       Error := Exception(ExcRecord.ExceptionObject); // Obtain exception type...
       MessageBox(0,PChar(Format('Error: %s',[Error.Message])),'Error',
                  MB_OK or MB_ICONSTOP or MB_TASKMODAL);
       ExcRecord.CurrentModuleOptions.ExceptionDialogType := edtNone; // Don't show exception dial
     end;
end;
```

5.3 ExceptionActionNotify event

5.3.1 How to use this event

EurekaLog includes an "ExceptionActionNotify" event for interaction with the user program when an exception is raised and is being handled by EurekaLog. This user-supplied routine is called at each of several points within the EurekaLog system.

The syntax of this event is:

```
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord 69;
                         EurekaAction: TEurekaActionType 75; var Execute: Boolean);
```

To use this event you must create a routine with the indicated parameters and assign it to the ExceptionActionNotify EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   ...
   ...  // Your code...
```

```
   ...
end;

begin
   // Assign ExceptionActionNotify variable to MyActionNotify procedure...
   ExceptionActionNotify := MyActionNotify;
end.
```

The user-supplied routine is called at each of nine action points within EurekaLog:

```
atShowingExceptionInfo (before): call made before showing exception information.
atShowedExceptionInfo  (after) : call made after showing exception information.
atSavingLogFile        (before): call made before saving Log File.
atSavedLogFile         (after) : call made after saving Log File.
atSendingEmail         (before): call made before sending Email.
atSentEmail            (after) : call made after sending Email.
atSendingWebMessage    (before): call made before sending Web message
atSentWebMessage       (after) : call made after sending Web message
atTerminating          (before): call made before terminating application (click
'Terminate' button).
```

If you set the *"Execute"* parameter to *False* in a **before** action, the standard EurekaLog action will not be executed when your routine returns control to EurekaLog. In **after** actions, the *"Execute"* parameter can be used to check if the action was performed successfully.

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnExceptionActionNotify 64 event.

5.3.2 Examples

Some examples:

```
// Show exception dialog?
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if EurekaAction = atShowingExceptionInfo then
     begin
       Execute := MessageBox(0,'Do you want to show error details?', 'Question',
                             MB_YESNO or MB_ICONQUESTION or MB_TASKMODAL)<>ID_YES;
     end;
end;


// Don't save Log File...
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if EurekaAction = atSavingLogFile then
     Execute := False;
end;


// Check that exception was correctly saved to Log File...
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if EurekaAction = atSavedLogFile then
```

```
    begin
      if not Execute then
        MessageBox(0,'Cannot write to Log File.', 'Error',
                   MB_OK or MB_ICONSTOP or MB_TASKMODAL);
    end;
end;


// Check Email sent correctly...
procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if EurekaAction=atSentEmail then
     begin
       if not Execute then
         MessageBox(0,'Could not send the Email.', 'Error',
                    MB_OK or MB_ICONSTOP or MB_TASKMODAL);
     end;
end;
```

5.4 ExceptionErrorNotify event

5.4.1 How to use this event

EurekaLog includes an "ExceptionErrorNotify" event for interaction with the user program when a raised exception is being handled by EurekaLog and an error occurrs.

The syntax of this event is:

```
procedure MyErrorNotify(EurekaExceptionRecord: TEurekaExceptionRecord 69;
                        EurekaAction: TEurekaActionType 75; var Retry: Boolean);
```

To use this event you must create a routine with the indicated parameters and assign it to the ExceptionErrorNotify EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyErrorNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                        EurekaAction: TEurekaActionType; var Retry: Boolean);
begin
   ...
   ...  // Your code...
   ...
end;

begin
   // Assign ExceptionErrorNotify variable to MyErrorNotify procedure...
   ExceptionErrorNotify := MyErrorNotify;
end.
```

The user-supplied routine is called if an error occurs during the processing of any of the following actions:

```
atSavedLogFile         (after) : call made after saving Log File.
atSentEmail            (after) : call made after sending Email.
atSentWebMessage       (after) : call made after sending Web message
```

If you set *"Retry"* to *True*, EurekaLog will retry the last action.

You can use this event to change some EurekaLog settings in accord with the computer configuration.

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnExceptionErrorNotify 64 event.

5.4.2 Examples

Some examples:

```
// Change email settings...
procedure MyErrorNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                        EurekaAction: TEurekaActionType; var Retry: Boolean);
begin
   // An error occurred when sending the email...
   if EurekaAction = atSentEmail then
     begin
       // Try to send the email via SMTP client.
       EurekaExceptionRecord.CurrentModuleOptions.SMTPHost := 'localhost';
       Retry := True; // Force retry of this action.
     end;
end;


// Change the log settings...
procedure MyErrorNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                        EurekaAction: TEurekaActionType; var Retry: Boolean);
begin
   // An error occurred while saving the Log-File...
   if EurekaAction = atSavedLogFile then
     begin
       // Try to send again after changing the Log-File path.
       EurekaExceptionRecord.CurrentModuleOptions.OutputPath := 'c:\';
       Retry := True; // Force to retry of this action.
     end;
end;
```

5.5 AttachedFilesRequest event

5.5.1 How to use this event

EurekaLog includes an "AttachedFilesRequest" event for interaction with the user program when a raised exception is handled by EurekaLog and the user want send custom files with the EurekaLog message (via Email and Web).

The syntax of this event is:

```
procedure MyAttachedFiles(EurekaExceptionRecord: TEurekaExceptionRecord 69; AttachedFiles: TString
```

To use this event you must create a routine with the indicated parameters and assign it to the AttachedFilesRequest EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyAttachedFiles(EurekaExceptionRecord: TEurekaExceptionRecord; AttachedFiles: TStrings);
begin
   ...
```

```
   ...  // Your code...
   ...
end;

begin
   // Assign AttachedFilesRequest variable to MyAttachedFiles procedure...
   AttachedFilesRequest := MyAttachedFiles;
end.
```

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnAttachedFilesRequest[64] event.

5.5.2 Examples

Some examples:

```
// Attach the "C:\Windows\Notepad.exe" and "C:\AutoExec.bat" files to the sending message.
procedure MyAttachedFiles(EurekaExceptionRecord: TEurekaExceptionRecord; AttachedFiles: TStrings);
begin
   AttachedFiles.Add('C:\Windows\Notepad.exe');
   AttachedFiles.Add('C:\AutoExec.bat');
end;


// Clear all attached files (screenshot last HTML page, ...) and add the "C:\Config.sys" file.
procedure MyAttachedFiles(EurekaExceptionRecord: TEurekaExceptionRecord; AttachedFiles: TStrings);
begin
   AttachedFiles.Clear; // Clear all EurekaLog attached files (as screenshot, last HTML page,...)
   AttachedFiles.Add('C:\Config.sys');
end;
```

5.6 CustomDataRequest event

5.6.1 How to use this event

EurekaLog includes a "CustomDataRequest" event for interaction with the user program when a raised exception is handled by EurekaLog and the user want add some custom information to the Log text.

The syntax of this event is:

```
procedure MyCustomData(EurekaExceptionRecord: TEurekaExceptionRecord[69]; DataFields: TStrings);
```

Use the DataFields list to add all the needed values using the form *"DataFields.Add ('FieldName=FieldValue')"*.

To use this event you must create a routine with the indicated parameters and assign it to the CustomDataRequest EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyCustomData(EurekaExceptionRecord: TEurekaExceptionRecord; DataFields: TStrings);
begin
   ...
   ...  // Your code...
   ...
end;
```

```
begin
   // Assign CustomDataRequest variable to MyCustomData procedure...
   CustomDataRequest := MyCustomData;
end.
```

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnCustomDataRequest[64] event.

5.6.2 Examples

Some examples:

```
// Add the "C:\MyData.txt" text file data to the Log data.
procedure MyCustomData(EurekaExceptionRecord: TEurekaExceptionRecord; DataFields: TStrings);
var
   TextFile: TStrings;
begin
   TextFiles := TStringList.Create;
   try
     TextFiles.LoadFromFile('C:\MyData.txt'); // Load the file content
     DataFields.Add('File=' + TextFiles.Text); // Add the file context to the Log data
   finally
     TextFiles.Free;
   end;
end;


// Add a global variable (MainSQL) to the Log data.

// Global declaration...
var
  MainSQL: string;

procedure MyCustomData(EurekaExceptionRecord: TEurekaExceptionRecord; DataFields: TStrings);
begin
   DataFields.Add('SQL=' + MainSQL); // Add the MainSQL variable to the Log data
end;
```

5.7 CustomWebFieldsRequest event

5.7.1 How to use this event

EurekaLog includes a "CustomWebFieldsRequest" event for interaction with the user program when a raised exception is handled by EurekaLog and the user want send a message to a Web server sending custom HTTP fields (via post) together the upload files.

The syntax of this event is:

```
procedure MyCustomFields(EurekaExceptionRecord: TEurekaExceptionRecord[69]; WebFields: TStrings);
```

Use the WebFields list to add all the needed Web fields using the form *"WebFields.Add ('FieldName=FieldValue')"*.

To use this event you must create a routine with the indicated parameters and assign it to the CustomWebFieldsRequest EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...
```

```
// This is a normal procedure (not a method)...
procedure MyCustomWebFields(EurekaExceptionRecord: TEurekaExceptionRecord; WebFields: TStrings)
begin
   ...
   ...  // Your code...
   ...
end;

begin
   // Assign CustomFieldRequest variable to MyCustomField procedure...
   CustomWebFieldsRequest := MyWebCustomFields;
end.
```

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnCustomWebFieldsRequest event.

5.7.2 Examples

Some examples:

```
// Add userName and Password fields to the sending HTML page.
procedure MyCustomFields(EurekaExceptionRecord: TEurekaExceptionRecord; WebFields: TStrings);
begin
   // First HTTP field (UserName)...
   WebFields.Add('UserName=MyUserName');

   // Second HTTP field (Password)...
   WebFields.Add('Password=xyz');
end;


// Add the User Email field to the sending HTML page.
procedure MyCustomFields(EurekaExceptionRecord: TEurekaExceptionRecord; FieldsName, FieldsValue: T
begin
   WebFields.Add('Email=myemail@email.com');
end;
```

5.8 PasswordRequest event

5.8.1 How to use this event

EurekaLog includes a "PasswordRequest" event for interaction with the user program when a raised exception is handled by EurekaLog and all the Debug data (Unit, Class and Procedure names) are saved as encrypted values.

This event allow to obtain the exact password required to can decrypt all the encrypted data, otherwise all the data will be store in the Log text in encrypted form (the software author will be decrypt this data using the decrypt capabilities of EurekaLog Viewer software).

The syntax of this event is:

```
procedure MyPasswordRequest(EurekaExceptionRecord: TEurekaExceptionRecord; var Password: String
```

To use this event you must create a routine with the indicated parameters and assign it to the PasswordRequest EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
```

```
procedure MyPasswordRequest(EurekaExceptionRecord: TEurekaExceptionRecord; var Password: String);
begin
   ...
   ...  // Your code...
   ...
end;

begin
   // Assign PasswordRequest variable to MyPasswordRequest procedure...
   PasswordRequest := MyPasswordRequest;
end.
```

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnPasswordRequest 64 event.

5.8.2 Examples

Some examples:

```
// Obtain the required password asking it to the final user.
procedure MyPasswordRequest(EurekaExceptionRecord: TEurekaExceptionRecord; var Password: String);
begin
   InputQuery('Request', 'Insert the BUG report password', Password);
end;


// Obtain the required password by configuration INI file.
procedure MyPasswordRequest(EurekaExceptionRecord: TEurekaExceptionRecord; var Password: String);
var
   INI: TIniFile;
begin
   INI := TiniFile.Create('Config.ini');
   try
     Password := INI.ReadString('Main', 'Password', '');
   finally
     INI.Free;
   end;
end;
```

5.9 CustomButtonClickNotify event

5.9.1 How to use this event

EurekaLog includes an "CustomButtonClickNotify" event that will be executed every time that an user will click on the "Customizable Help Button 16".

The syntax of this event is:

```
procedure MyCustomButtonClickNotify(EurekaExceptionRecord: TEurekaExceptionRecord; var CloseDialog
```

To use this event you must create a routine with the indicated parameters and assign it to the CustomButtonClickNotify EurekaLog variable, as shown in the following example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

// This is a normal procedure (not a method)...
procedure MyCustomButtonClickNotify(EurekaExceptionRecord: TEurekaExceptionRecord; var CloseDialog
begin
   ...
```

```
   ...  // Your code...
   ...
end;

begin
   // Assign CustomButtonClickNotify variable to CustomButtonClickNotify procedure...
   CustomButtonClickNotify := MyCustomButtonClickNotify;
end.
```

Note: to simplify management of your EurekaLog events, you can use the TEurekaLog.OnCustomButtonClickNotify 64 event.

5.9.2 Examples

Some examples:

```
// Close the dialog.
procedure MyCustomButtonClickNotify(EurekaExceptionRecord: TEurekaExceptionRecord; var CloseDialog
begin
   CloseDialog := True;
end;


// Show a MessageBox.
procedure MyCustomButtonClickNotify(EurekaExceptionRecord: TEurekaExceptionRecord; var CloseDialog
begin
   MessageBox(0, 'You have clicked on the HELP button!',
     'Information', MB_OK or MB_ICONINFORMATION or MB_TASKMODAL);
end;
```

Part

6 Generic functions

6.1 StandardEurekaNotify function

EurekaLog includes a function called *StandardEurekaNotify*, that enables you to call EurekaLog "manually".

Syntax of this function is as follows:

```
function StandardEurekaNotify(Obj: TObject; Addr: pointer): Boolean;
```

When EurekaLog is active the Result is *True*, *False* otherwise (i.e. when the *ExceptionLog.pas* unit is included in your project but you have disabled EurekaLog from the "Project/EurekaLog Options..." IDE menu).

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   ok := StandardEurekaNotify(ExceptObject, ExceptAddr);
   ...
end;
```

6.2 IsEurekaLogInstalled function

EurekaLog includes a function named *IsEurekaLogInstalled*, that enables you to check if EurekaLog is installed in your application.

Syntax of this function is as follows:

```
function IsEurekaLogInstalled: Boolean;
```

When EurekaLog is active the Result is *True*, *False* otherwise (when the *ExceptionLog.pas* unit is included in your project but you have disabled EurekaLog from the "Project/Exceptions Log Options..." IDE menù).

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   ok := IsEurekaLogInstalled;
   ...
end;
```

6.3 CurrentEurekaLogOptions function

EurekaLog includes a function named *CurrentEurekaLogOptions*, that provides access to the EurekaLog options of the current module.

Syntax of this function is as follows:

```
function CurrentEurekaLogOptions: TEurekaModuleOptions[69];
```

When EurekaLog is active the Result is the EurekaLog options of the calling module, otherwise this function raises an exception (you can first check that EurekaLog is installed by calling the IsEurekaLogInstalled[48] function).

Note:
Don't use this function within EurekaLog events. Use instead the TEurekaExceptionRecord.CurrentModuleOptions[69] parameter.

Example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

begin
   ...
   CurrentEurekaLogOptions.EMailAddresses := 'bugs@eurekalog.com';
   CurrentEurekaLogOptions.OutputPath := ExtractFilePath(ParamStr(0));
   ...
end;
```

6.4 ForceApplicationTermination function

EurekaLog includes a function named *ForceApplicationTermination*, that enables you to force your application to terminate/restart after handling the current exception.

Syntax of this function is as follows:

```
function ForceApplicationTermination(TrmType: TTerminateBtnOperation[80]): Boolean;
```

Use the "TrmType" parameter to choose the termination type (simple termination / termination with restart).

When EurekaLog is active the Result is True, False otherwise (when the ExceptionLog.pas unit is included in your project but you have disabled EurekaLog from the "Project/Exceptions Log Options..." IDE menù).

Example:

```
uses ExceptionLog, ECore; // The required units...

begin
   ...
   ok := ForceApplicationTermination(tbRestart); // Force the application to restart.
   ...
end;
```

6.5 StandardEurekaError function

EurekaLog includes a function named *StandardEurekaError*, that enables you to call EurekaLog "manually" simulationg an exception.

Syntax of this function is as follows:

```
function StandardEurekaError(const Error: string): Boolean;
```

When EurekaLog is active the Result is *True*, otherwise it returns *False* (when the *ExceptionLog.pas* unit is included in your project but you have EurekaLog disabled in the "Project/Exceptions Log Options..." IDE menu).

With this function you can obtain the current call-stack without raising an exception.

This function simulates the raising of an EEurekaLogGeneralError 80 exception with the Exception.Message = Error.

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   ok := StandardEurekaError('General error!');
   ...
end;
```

6.6 GetLastEurekaLogErrorCode function

EurekaLog includes a function named *GetLastEurekaLogErrorCode*, that enables you to obtain the last internal EurekaLog error code.

Syntax of this function is as follows:

```
function GetLastEurekaLogErrorCode: TEurekaLogErrorCode 80;
```

You should use this function in the EurekaLog ExceptionActionNotify 36 event to check if for errors in the email sending procedure (or other).

Example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if (EurekaAction = atSentEmail) and (GetLastEurekaLogErrorCode = eeEmailMAPIError) then
     begin
       MessageBox(0,'Cannot connect with the email client using the MAPI protocol.', 'Error',
                  MB_OK or MB_ICONERROR);
     end;
end;
```

6.7 GetLastEurekaLogErrorMsg function

EurekaLog includes a function named *GetLastEurekaLogErrorMsg*, that enables you to obtain the last internal EurekaLog error message.

Syntax of this function is as follows:

```
function GetLastEurekaLogErrorMsg: string;
```

You should use this function in the EurekaLog ExceptionActionNotify 36 event to check if for errors in the email sending procedure (or other).

Example:

```
uses ExceptionLog, ECore, ETypes; // The required units...

procedure MyActionNotify(EurekaExceptionRecord: TEurekaExceptionRecord;
                         EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   if (EurekaAction = atSentEmail) and
      (GetLastEurekaLogErrorCode in [eeEmailMAPIError, eeEmailShellError, eeEmailSMTPError]) then
     begin
       MessageBox(0,PChar(GetLastEurekaLogErrorMsg), 'Email error', MB_OK or MB_ICONERROR);
     end;
end;
```

6.8 SetEurekaLogInThread procedure

EurekaLog includes a procedure called *SetEurekaLogInThread*, that enables you to activate/ deactivate EurekaLog in a specified thread.

Syntax of this function is as follows:

```
procedure SetEurekaLogInThread(ThreadID: DWord; Activate: Boolean);
```

The *ThreadID* parameter indicates the Thread ID and *Activate* indicates the EurekaLog state (Active/Disactive).

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   SetEurekaLogInThread(GetCurrentThreadID, False); // Disable EurekaLog in the current Thread.
   ...
end;
```

6.9 IsEurekaLogActiveInThread function

EurekaLog includes a function called *IsEurekaLogActiveInThread*, that enables you to check the EurekaLog state in a specified thread.

Syntax of this function is as follows:

```
function IsEurekaLogActiveInThread(ThreadID: DWord): Boolean;
```

The *ThreadID* parameter indicates the Thread ID.

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
```

```
   OK := IsEurekaLogActiveInThread(GetCurrentThreadID); // Get the state of EurekaLog in current T
   ...
end;
```

6.10 SetEurekaLogState procedure

EurekaLog includes a procedure called *SetEurekaLogState*, that enables you to activate/ deactivate EurekaLog.

Syntax of this function is as follows:

```
procedure SetEurekaLogState(Activate: Boolean);
```

The *Activate* parameter indicates the EurekaLog state (Active/Disactive).

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   SetEurekaLogState(False); // Disable EurekaLog.
   ...
end;
```

6.11 IsEurekaLogActive function

EurekaLog includes a function called *IsEurekaLogActive*, that enables you to check the EurekaLog state.

Syntax of this function is as follows:

```
function IsEurekaLogActive: Boolean;
```

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   OK := IsEurekaLogActive; // Get the state of EurekaLog.
   ...
end;
```

6.12 EurekaLog version constants

EurekaLog includes two constants, *EurekaLogCurrentVersion* (Word) and *EurekaLogVersion* (String) that enables you to check the current EurekaLog version.

Example:

```
uses EConsts; // The required unit...

begin
   ...
   // Write 4.5.0 for the EurekaLog 4.5.0 version.
```

```
   WriteLn(EurekaLogVersion);

   // Write 'OK' if EurekaLog version if 4.2.0 or higher.
   if (EurekaLogCurrentVersion >= 420) then WriteLn('OK');
   ...
end;
```

6.13 SaveScreenshot procedure

EurekaLog includes a function called *SaveScreenshot*, that enables you to save the current screenshot to a file (.PNG format). The saving options are gets by the EurekaLog options.

Syntax of this function is as follows:

```
procedure SaveScreenshot(const Filename: string);
```

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   SaveScreenshot('C:\Screenshots\MyFile.png');
   ...
end;
```

6.14 SaveScreenshotToStream procedure

EurekaLog includes a function called *SaveScreenshotToStream*, that enables you to save the current screenshot to a stream (.PNG format). The saving options are gets by the EurekaLog options.

Syntax of this function is as follows:

```
procedure SaveScreenshotToStream(Stream: TStream);
```

Example:

```
uses ExceptionLog; // The required unit...

var
   FS: TFileStream;
begin
   FS := TFileStream.Create('Screenshot.png', fmCreate);
   try
     SaveScreenshotToStream(FS);
   finally
     FS.Free;
   end;
end;
```

6.15 IsEurekaLogModule function

EurekaLog includes a function called *IsEurekaLogModule*, that enables you to check if a module has been compiled with the EurekaLog support.

Syntax of this function is as follows:

```
function IsEurekaLogModule(HModule: THandle): Boolean;
```

Example:

```
uses ExceptionLog; // The required unit...

procedure LoadLib;
var
   Lib: THandle;
   Res: Boolean;
begin
   Lib := LoadLibrary('C:\Library.dll');
   if (Lib <> 0) then
   try
     Res := IsEurekaLogModule(Lib);
     if (Res) then
       MessageBox(0, 'The "Lybrary.dll" module has been compiled WITH the EurekaLog support.',
        'Information', MB_OK or MB_ICONINFORMATION)
     else
       MessageBox(0, 'The "Lybrary.dll" module has been compiled WITHOUT the EurekaLog support.',
       'Information', MB_OK or MB_ICONINFORMATION)
   finally
     FreeLibrary(Lib);
   end;
end;
```

6.16 GetEurekaLogModuleVersion function

EurekaLog includes a function called *GetEurekaLogModuleVersion*, that enables you to obtain the EurekaLog version used to compile a specified module (0 if the module it isn't compiled with EurekaLog, 400 if is compiled with EurekaLog 4.0.0, 512 if is compiled with EurekaLog 5.1.2, ...).

Syntax of this function is as follows:

```
function GetEurekaLogModuleVersion(HModule: THandle): Word;
```

Example:

```
uses ExceptionLog; // The required unit...


// This function returns the EurekaLog version used to compile a specified file...
function GetEurekaLogFileVersion(const FileName: string): Word;
var
   Module: THandle;
begin
   Result := 0;
   // Load the file to obtain its Handle...
   Module := LoadLibrary(PChar(FileName));
   if (Module <> 0) then
   try
     Result := GetEurekaLogModuleVersion(Module);
   finally
     // Unload the file...
     FreeLibrary(Module)
```

```
   end;
end;
```

6.17 GetCompilationDate function

EurekaLog includes a function called *GetCompilationDate*, that enables you to obtain the compilation date of a specified module (in local or global-GMT time coordinates).
Syntax of this function is as follows:

```
function GetCompilationDate(HModule: THandle; LocalTime: Boolean; var Date: TDateTime): Boolean;
```

The first parameter indicate the module handle (HInstance for the current module), the second indicate the result format and the last parameter indicate the result value. The function returns True if the required module was compiled with EurekaLog and False otherwise.

Example:

```
uses ExceptionLog; // The required unit...

var
   Date: TDateTime;
begin
   ...
   if GetCompilationDate(HInstance, True, Date) then
     MessageBox(0, PChar(FormatDateTime('dd/mm/yyyy hh:nn:ss', Date)), '', 0);
   ...
end;
```

6.18 GenerateHTML function

EurekaLog includes a function called *GenerateHTML*, that enables you to convert a custom plain text in a formatted HTML code (the formatting process uses the HTML layout defined in the EurekaLog options).

NOTE: You can use this function to show the EurekaLog error page in unsupported Web applications types 95.

Syntax of this function is as follows:

```
function GenerateHTML(const Text: string; AddOKButton: Boolean): string;
```

Example:

```
uses ExceptionLog; // The required unit...

var
   HTMLCode: string;
begin
   ...
   HTMLCode := GenerateHTML('Custom text...', True {Add a JavaScript 'back' button at the page bot
   ...
end;
```

6.19 SetCustomErrorMessage procedure

EurekaLog includes a procedure called *SetCustomErrorMessage*, that enables you to set a custom error message to display in the Exception Dialog instead of exception message *(only when the dialog is not displayed in detailed mode)*.

NOTE: Use an empty string to remove any pevious custom error message *(Example: SetCustomErrorMessage(''));)*.

Syntax of this function is as follows:

```
procedure SetCustomErrorMessage(const Value: string);
```

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   SetCustomErrorMessage('An error has occurred.');
   ...
end;
```

6.20 SetProxyServer procedure

EurekaLog includes a procedure called *SetProxyServer*, that enables you to set a custom Proxy server. By default EurekaLog use the System Internet Settings to get the Proxy configuration but using this procedure you can force the use of custom Proxy settigns.

Syntax of this function is as follows:

```
procedure SetProxyServer(const Server: string; Port: Word);
```

Example:

```
uses EWebTools; // The required unit...

begin
   ...
   SetProxyServer('www.my-proxy.com', 8080);
   ...
end;
```

6.21 SetProxyAuthenticationData procedure

EurekaLog includes a procedure called *SetProxyAuthenticationData*, that enables you to set the Proxy UserID and Password.

Syntax of this function is as follows:

```
procedure SetProxyAuthenticationData(const UserID, Password: string);
```

Example:

```
uses EWebTools; // The required unit...

begin
   ...
   SetProxyAuthenticationData('proxy_username', 'proxy_password');
   ...
end;
```

6.22 SetFTPPassiveMode procedure

EurekaLog includes a procedure called *SetFTPPassiveMode*, that enables you to activate/ deactivate the FTP PASV/Passive mode.

Syntax of this function is as follows:

```
procedure SetFTPPassiveMode(Activate: Boolean);
```

The *Activate* parameter indicates the FTP PASV/Passive (Active/Disactive).

Example:

```
uses EWebTools; // The required unit...

begin
   ...
   SetFTPPassiveMode(False); // Disable the FTP PASV mode.
   ...
end;
```

6.23 GetLastExceptionAddress function

EurekaLog includes a function called *GetLastExceptionAddress*, that enables you to obtain the memory address of the last raised exception.

Syntax of this function is as follows:

```
function GetLastExceptionAddress: Pointer;
```

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   WriteLn(Format('Last Exception address: %p', [GetLastExceptionAddress]))
   ...
end;
```

6.24 GetLastExceptionObject function

EurekaLog includes a function called *GetLastExceptionObject*, that enables you to obtain the object instance (*TObject*) of the last raised exception (only if valid too, otherwise *nil*).

Syntax of this function is as follows:

```
function GetLastExceptionObject: TObject;
```

Example:

```
uses ExceptionLog; // The required unit...

begin
   ...
   if (GetLastExceptionObject <> nil) then
     WriteLn(Format('Last Exception class name: "%s"', [GetLastExceptionObject.ClassName]))
   else
     WriteLn('Unknown last exception class name!');
   ...
end;
```

6.25 ShowLastExceptionData procedure

EurekaLog includes a procedure called *ShowLastExceptionData*, that enables you to display the data of the last raised exception.

This procedure is usefully if used in a *"re-raised block"* (*a raise command in an except/end block*) without lost the original exception line.

Syntax of this procedure is as follows:

```
procedure ShowLastExceptionData;
```

Example (with raise command):

```
uses ExceptionLog; // The required unit...

begin
   ...
   try
     raise Exception.Create('Original exception!');
   except
     raise; // <-- WARNING - Will be displayed this line as exception line,
            //               losing the original exception line!!!
   end;
   ...
end;
```

Example (with ShowLastExceptionData procedure):

```
uses ExceptionLog; // The required unit...

begin
   ...
   try
     raise Exception.Create('Original exception!'); // <-- Will be displayed this line as exceptio
   except
     ShowLastExceptionData;
     Abort;
   end;
   ...
```

```
end;
```

6.26 EurekaLogSendEmail function

EurekaLog includes a function called *EurekaLogSendEmail*, that enables you to send en email to one o more recipient using the default email client installed on your PC.

Syntax of this function is as follows:

```
function EurekaLogSendEmail(const AMailTo, ASubject, ABody, AAttachments: string): Boolean;
```

Note:
You can insert more recipients and attachments, separating them with a ';' or a ',' char.

Example:

```
uses ExceptionLog; // The required unit...

var
   Sent: boolean;
begin
   Sent := EurekaLogSendEmail('first@email.com;second@supportemail.com', 'Test email',
     'This is a test email!', 'C:\Log.txt;C:\Windows\notepad.exe');
end;
```

6.27 CallStackToStrings procedure

EurekaLog includes a procedure called *CallStackToStrings*, that enables you to obtain a textual representation of a call-stack.

This procedure is usefully to work with all the EurekaLog routines able to returns a call-stack type (TEurekaStackList 72).

Syntax of this procedure is as follows:

```
procedure CallStackToStrings(CallStack: TEurekaStackList; Strings: TStrings);
```

Note:
The text output format is full compatible with the EurekaLog log-file (.ELF) call-stack format.

Example:

```
uses ExceptionLog; // The required unit...

var
   CallStackList: TStringList;
   CallStack: TEurekaStackList;
begin
   CallStackList := TStringList.Create;
   try
     CallStack := GetCurrentCallStack;
     try
       CallStackToStrings(CallStack, CallStackList);
       // ...
       // Use the CallStackList here...
       // ...
```

```
    finally
      CallStack.Free;
    end;
  finally
    CallStackList.Free;
  end;
end;
```

6.28 GetCurrentCallStack function

EurekaLog includes a function named *GetCurrentCallStack*, that enables you to obtain the current call-stack.

Syntax of this function is as follows:

```
function GetCurrentCallStack: TEurekaStackList;
```

Example:

```
uses ExceptionLog; // The required unit...

var
  CallStackList: TStringList;
  CallStack: TEurekaStackList;
begin
  CallStackList := TStringList.Create;
  try
    CallStack := GetCurrentCallStack;
    try
      CallStackToStrings(CallStack, CallStackList);
      // ...
      // Use the CallStackList here...
      // ...
    finally
      CallStack.Free;
    end;
  finally
    CallStackList.Free;
  end;
end;
```

6.29 GetLastExceptionCallStack function

EurekaLog includes a function called *GetLastExceptionCallStack*, that enables you to obtain the last Exception Call-Stack (it works with handled and unhandled exceptions).

Syntax of this function is as follows:

```
function GetLastExceptionCallStack: TEurekaStackList;
```

Example:

```
uses ExceptionLog; // The required unit...

var
  CallStackList: TStringList;
```

```
   CallStack: TEurekaStackList;
begin
   CallStackList := TStringList.Create;
   try
     CallStack := GetLastExceptionCallStack;
     try
       CallStackToStrings(CallStack, CallStackList);
       // ...
       // Use the CallStackList here...
       // ...
     finally
       CallStack.Free;
     end;
   finally;
     CallStackList.Free;
   end;
end;
```

6.30 GetCallStackByLevels function

EurekaLog includes a function called *GetCallStackByLevels*, that enables you to obtain only a custom subset of the current call-stack. You can choose what levels obtains (the level=0 indicate the current running routine, the level=1 the caller of the current routine, ...).

Syntax of this function is as follows:

```
function GetCallStackByLevels(StartLevel, LevelsNumber: Integer): TEurekaStackList;
```

Example:

```
uses ExceptionLog; // The required unit...

var
   CallStackList: TStringList;
   CallStack: TEurekaStackList;
begin
   CallStackList := TStringList.Create;
   try
     CallStack := GetCallStackByLevels(0, 99);
     try
       CallStackToStrings(CallStack, CallStackList);
       // ...
       // Use the CallStackList here...
       // ...
     finally
       CallStack.Free;
     end;
   finally;
     CallStackList.Free;
   end;
end;
```

6.31 GetSourceInfoByAddr function

EurekaLog includes a function called *GetSourceInfoByAddr*, that enables you to obtain debug information of a specific memory address (ex: the unit name or the line number).

Syntax of this function is as follows:

```
function GetSourceInfoByAddr(Addr: DWord; DebugInfo: PEurekaDebugInfo): Boolean;
```

Example:

```
uses ExceptionLog; // The required unit...

function GetSourceCodeStringFromAddr(Addr: DWord): string;
var
  DebugInfo: TEurekaDebugInfo;
begin
  Result := '';

  if GetSourceInfoByAddr(Addr, @DebugInfo) then
  begin
    Result := Format('%s.%s.%s.%d[%d]', [
      DebugInfo.UnitName, DebugInfo.ClassName,
      DebugInfo.ProcedureName, DebugInfo.Line,
      DebugInfo.ProcOffsetLine]);
  end;
end;
```

Part VII

7 TEurekaLog component

7.1 How to use this component

EurekaLog install the TEurekaLog component into the EurekaLog component palette.

This component has seven events:

- OnExceptionNotify a wrapper for the ExceptionNotify 36 EurekaLog event (called when EurekaLog traps an UNHANDLED exception).
- OnHandledExceptionNotify a wrapper for the HandledExceptionNotify 37 EurekaLog event (called when EurekaLog traps an HANDLED exception).
- OnExceptionActionNotify a wrapper for the ExceptionActionNotify 38 EurekaLog event.
- OnExceptionErrorNotify a wrapper for the ExceptionErrorNotify 40 EurekaLog event.
- OnAttachedFilesRequest a wrapper for the AttachedFilesRequest 41 EurekaLog event
- OnCustomDataRequest a wrapper for the CustomDataRequest 42 EurekaLog event.
- OnCustomWebFieldsRequest a wrapper for the CustomWebFieldsRequest 43 EurekaLog event.
- OnPasswordRequest a wrapper for the PasswordRequest 44 EurekaLog event.
- OnCustomButtonClickNotify a wrapper for the CustomButtonClickNotify 45 EurekaLog event.

Warning:

You can use this component to simplify management of your EurekaLog events, but it's more reliable use the original events because TEurekaLog component is created only when its parent form is created did not allowing to handled EurekaLog events before its creation.

See the "Examples" 64 topic for further details.

7.2 Examples

Some examples:

```
uses ECore, ETypes; // The required units...

// OnExceptionNotify example...
procedure TForm1.EurekaLog1ExceptionNotify(
   EurekaExceptionRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   // Don't save the Log File...
   EurekaExceptionRecord.CurrentModuleOptions.SaveLogFile := False;
end;
```

See the "ExceptionNotify-Examples" 36 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnHandledExceptionNotify example...
procedure TForm1.EurekaLog1HandledExceptionNotify(
   EurekaExceptionRecord: TEurekaExceptionRecord; var Handled: Boolean);
begin
   // Don't save the Log File...
   EurekaExceptionRecord.CurrentModuleOptions.SaveLogFile := False;
end;
```

See the "ExceptionNotify-Examples" 36 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnExceptionActionNotify example...
procedure TForm1.EurekaLog1ExceptionActionNotify(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   EurekaAction: TEurekaActionType; var Execute: Boolean);
begin
   // Show exception dialog?
   if EurekaAction = atShowingExceptionInfo then
     begin
       Execute := MessageBox(0,'Do you want to show error details?', 'Question',
                    MB_YESNO or MB_ICONQUESTION or MB_TASKMODAL)<>ID_YES;
     end;
end;
```

See the "ExceptionActionNotify-Examples" 41 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnExceptionErrorNotify example...
procedure TForm1.EurekaLog1ErrorNotify(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   EurekaAction: TEurekaActionType; var Retry: Boolean);
begin
   // An error occurred when sending the email...
   if EurekaAction = atSentEmail then
     begin
       // Try to send the email via SMTP client.
       EurekaExceptionRecord.CurrentModuleOptions.SMTPHost := 'localhost';
       Retry := True; // Force retry of this action.
     end;
end;
```

See the "ExceptionErrorNotify-Examples" 41 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnAttachedFilesRequest example...
```

```
procedure TForm1.EurekaLog1AttachedFilesRequest(
   EurekaExceptionRecord: TEurekaExceptionRecord;
  AttachedFiles: TStrings);
var
   Path: string;
begin
   // Attach a file chose by user...
   Path := InputBox('Request', 'Insert the path of the file to send', '');
   AttachedFiles.Add(Path);
end;
```

See the "AttachedFilesRequest-Examples" 42 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnCustomDataRequest example...
procedure TForm1.EurekaLog1CustomDataRequest(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   DataFields: TStrings);
begin
   // Add custom data to the Log text...
   DataFields.Add('Field=My custom data...';
end;
```

See the "CustomDataRequest-Examples" 43 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnCustomFieldsRequest example...
procedure TForm1.EurekaLog1CustomWebFieldsRequest(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   WebFields: TStrings);
begin
   // Send the software identification ID to the Web server
   FieldsName.Add('Software_ID');
   FieldsValue.Add('F0E0D0');
end;
```

See the "CustomFieldsRequest-Examples" 44 topic for further details.

```
uses ECore, ETypes; // The required units...

// OnPasswordRequest example...
procedure TForm1.EurekaLog1PasswordRequest(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   var Password: string);
begin
   // Require the password to the user...
   InputQuery('Request', 'Insert the BUG report password', Password);
end;
```

See the "PasswordRequest-Examples" 45 topic for further details.

```
uses ECore, ETypes; // The required units...

// Close the dialog.
procedure TForm1.EurekaLog1CustomButtonClickNotify(
   EurekaExceptionRecord: TEurekaExceptionRecord;
   var CloseDialog: Boolean);
begin
   CloseDialog := True;
end;
```

See the "CustomButtonClickNotify-Examples" 46 topic for further details.

Part VII

8 Types

8.1 TEurekaExceptionRecord type

This one is the structure of TEurekaExceptionRecord:

Unit ExceptionLog.

```
TEurekaExceptionRecord = packed record
  ExceptionObject:      TObject;
  ExceptionAddress:     Pointer;
  ExceptionThreadID:    DWord;
  Win32ExceptionRecord: PExceptionRecord;
  Win32Context:         PContext;
  LogText:              string;
  ModulesList:          TEurekaModulesList 74;
  ProcessesList:        TEurekaProcessesList 74;
  CallStack:            TEurekaStackList 72;
  CurrentModuleOptions: TEurekaModuleOptions 69;
end;
```

Note: if the ExceptionObject is an Exception than is possible change its "Message" property text to show a most friendly error message, but into the log-file will be stored the original Exception Message.

8.2 TEurekaModuleOptions type

This is the TEurekaModuleOptions type:

Unit ECore.

```
TEurekaModuleOptions = class(TObject)
...
public
  // Methods...
  constructor Create(ModuleName: string ['']; SaveSharedData: Boolean [False]);
  procedure   Assign(Source: TEurekaModuleOptions 69);
  procedure   SetToDefaultOptions;
  procedure   SaveToStream(Stream: TStream);
  procedure   LoadFromStream(Stream: TStream);
  procedure   SaveToFile(FileName: string);
  procedure   LoadFromFile(FileName: string);
  procedure   SaveCustomizedTextsToFile(const FileName: string);
  procedure   LoadCustomizedTextsFromFile(const FileName: string);
  // Properties...
  property ModuleName:                          string ...;
  property SMTPFrom:                            string ...;
  property SMTPHost:                            string ...;
  property SMTPPort:                            Word ...;
  property SMTPUserID:                          string ...;
  property SMTPPassword:                        string ...;
  property FreezeActivate:                      Boolean ...;
  property FreezeTimeout:                       Integer ...;
  property ExceptionsFilters:                   TEurekaExceptionsFilters 71 ...;
  property CustomizedTexts[Index: TMessageType 76]: string ...;
  property TextsCollection:                     string ...;
  property ActivateLog:                         Boolean ...;
```

```
    property SaveLogFile:                       Boolean ...;
    property ActivateHandle:                    Boolean ...;
    property ForegroundTab:                     TForegroundType[76] ...;
    property AppendLogs:                        Boolean ...;
    property TerminateBtnOperation:             TTerminateBtnOperation[80] ...;
    property LogOptions:                        TLogOptions[79] ...;
    property ShowOptions:                       TShowOptions[79] ...;
    property ErrorsNumberToSave:                Integer ...;
    property ErrorsNumberToShowTerminateBtn:    Integer ...;
    property OutputPath:                        string ...;
    property EMailAddresses:                    string ...;
    property EMailSubject:                      string ...;
    property EMailMessage:                      string ...;
    property EMailSendMode:                     TEmailSendMode[80] ...;
    property WebSendMode:                       TWebSendMode[80] ...;
    property WebURL:                            string ...;
    property WebUserID:                         string ...;
    property WebPassword:                       string ...;
    property WebPort:                           Integer ...;
    property CommonSendOptions:                 TCommonSendOptions[78] ...;
    property AttachedFiles:                     string ...;
    property ExceptionDialogType:               TExceptionDialogType[71] ...;
    property ExceptionDialogOptions:            TExceptionDialogOptions[77] ...;
    property AutoCloseDialogSecs:               Integer ...;
    property SupportURL:                        string ...;
    property HTMLLayout:                        string ...;
    property AutoCrashOperation:                TTerminateBtnOperation[80] ...;
    property AutoCrashNumber:                   Integer ...;
    property AutoCrashMinutes:                  Integer ...;
    property CallStackOptions:                  TCallStackOptions[78] ...;
    property BehaviourOptions:                  TBehaviourOptions[78] ...;
    property LeaksOptions:                      TLeaksOptions[72] ...;
    property ProxyURL:                          string ...;
    property ProxyUserID:                       string ...;
    property ProxyPassword:                     string ...;
    property ProxyPort:                         Word ...;
    property TrakerUserID:                      string ...;
    property TrakerPassword:                    string ...;
    property TrakerAssignTo:                    string ...;
    property TrakerProject:                     string ...;
    property TrakerCategory:                    string ...;
    property TrakerTrialID:                     string ...;
    property ZipPassword:                       string ...;
    property CompiledFileOptions:               TCompiledFileOptions[72] ...;
    property ReproduceText:                     string ...;
  end;
```

8.3 TFilterExceptionType type

This is the TFilterExceptionType type:

Unit ECore.

```
  TFilterExceptionType = (fetAll, fetHandled, fetUnhandled);
```

8.4 TFilterDialogType type

This is the TFilterDialogType type:

Unit ECore.

```
  TFilterDialogType = (fdtNone, fdtUnchanged, fdtMessageBox, fdtMSClassic,
fdtEurekaLog);
```

8.5 TFilterHandlerType type

This is the TFilterHandlerType type:

Unit ECore.

```
  TFilterHandlerType = (fhtNone, fhtRTL, fhtEurekaLog);
```

8.6 TFilterActionType type

This is the TFilterActionType type:

Unit ECore.

```
  TFilterActionType = (fatNone, fatTerminate, fatRestart);
```

8.7 TEurekaExceptionFilter type

This is the TEurekaProcessInfo type:

Unit ECore.

```
  TEurekaExceptionFilter = packed record
    Active:             Boolean;
    ExceptionClassName: string;
    ExceptionMessage:   string;
    ExceptionType:      TFilterExceptionType[70];
    DialogType:         TFilterDialogType[70];
    HandlerType:        TFilterHandlerType[71];
    ActionType:         TFilterActionType[71];
  end;
  PEurekaExceptionFilter = ^TEurekaExceptionFilter;
```

8.8 TEurekaExceptionsFilters

This is the TEurekaStackList type:

Unit ExceptionLog.

```
  TEurekaExceptionsFilters = class(TList)
  ...
  public
    ...
    property Items[Index: Integer]: PEurekaExceptionFilter[71] read GetItem; default;
  end;
```

8.9 TExceptionDialogType

This is the TExceptionDialogType type:

Unit ECore.

```
  TExceptionDialogType = (edtNone, edtMessageBox, edtMSClassic, edtEurekaLog);
```

8.10 TLeaksOptions

This is the TLeaksOptions type:

Unit ECore.

```
TLeaksOptions = set of TLeaksOption;
```

8.11 TLeaksOption

This is the TLeaksOption type:

Unit ECore.

```
TLeaksOption = (loCatchLeaks, loGroupsSonLeaks, loHideBorlandLeaks,
                loFreeAllLeaks, loCatchLeaksExceptions);
```

8.12 TCompiledFileOptions

This is the TCompiledFileOptions type:

Unit ECore.

```
TCompiledFileOptions = set of TCompiledFileOption;
```

8.13 TCompiledFileOption

This is the TCompiledFileOption type:

Unit ECore.

```
TCompiledFileOption = (cfoReduceFileSize, cfoCheckFileCorruption);
```

8.14 TEurekaStackList type

This is the TEurekaStackList type:

Unit ExceptionLog.

```
TEurekaStackList = class(TList)
...
public
  ...
  property Items[Index: Integer]: PEurekaDebugInfo 72 read GetItem; default;
end;
```

8.15 TEurekaDebugInfo type

This is the PEurekaDebugInfo type:

Unit ExceptionLog.

```
TEurekaDebugInfo = packed record
  DebugDetail:    TEurekaDebugDetail 73;
  ModuleInfo:     PEurekaModuleInfo 73;
  ThreadID:       DWord;
  RunningThread:  Boolean;
```

```
    ErrorLine:      Boolean;
    IsALeak:        Boolean;
    LeakType:       string;
    LeakSize:       DWord;
    LeakCount:      DWord;
    Addr:           DWord;
    UnitName:       string;
    ClassName:      string;
    ProcedureName:  string;
    Line:           DWord;
    ProcOffsetLine: DWord;
  end;
  PEurekaDebugInfo = ^TEurekaDebugInfo;
```

8.16 TEurekaDebugDetail type

This is the TEurekaDebugDetail type:

Unit ExceptionLog.

```
  TEurekaDebugDetail = (ddNone, ddModule, ddProcedure, ddUnitAndProcedure,
ddSourceCode);
```

8.17 TEurekaModuleInfo type

This is the TEurekaModuleInfo type:

Unit ExceptionLog.

```
  TEurekaModuleInfo = packed record
    Handle:                 THandle;
    Name:                   string;
    Description:            string;
    Version:                string;
    Size:                   DWord;
    FunctionsList:          TEurekaFunctionsList[75];
    ModuleType:             TEurekaModuleType[75];
    ExtraInformation:       TEurekaExtraInformation[75];
    OtherDebugData:         TELDebugInfoSource;
    LastModified:           TDateTime;
    EncryptPassword:        string;
    IsValidEncryptPassword: Boolean;
  end;
  PEurekaModuleInfo = ^TEurekaModuleInfo;
```

8.18 TELDebugInfoSource type

This is the TELDebugInfoSource type:

Unit EDebug.

```
   TELDebugInfoSource = class(TObject)
  public
    constructor Create(AModule: HMODULE); virtual;
    function    ValidData: Boolean; virtual; abstract;
    function    GetLocationInfo(const Addr: Pointer; var Info: TELLocationInfo[74]): Boolean; virtu
    property    Module: HMODULE read FModule;
    property    FileName: TFileName read GetFileName;
  end;
```

8.19 TELLocationInfo type

This is the TELLocationInfo type:

Unit EDebug.

```
TELLocationInfo = record
  Address:             Pointer;            // Error address
  UnitName:            string;             // Name of unit
  ProcedureName:       string;             // Procedure name
  ProcOffsetLine:      Integer;            // Offset from Procedure line and Address line
  LineNumber:          Integer;            // Line number
  SourceName:          string;             // Module file name
  DebugInfo:           TELDebugInfoSource[73];  // Location object
end;
```

8.20 TEurekaModulesList type

This is the TEurekaModuleList type:

Unit ExceptionLog.

```
TEurekaModulesList = class(TList)
...
public
  ...
  property Items[Index: Integer]: PEurekaModuleInfo[73] read GetItem; default;
end;
```

8.21 TEurekaProcessInfo type

This is the TEurekaProcessInfo type:

Unit ExceptionLog.

```
TEurekaProcessInfo = packed record
  ProcessID:   DWord;
  Name:        string;
  Description: string;
  Version:     string;
  Priority:    DWord;
  Memory:      DWord;
  Threads:     DWord;
end;
PEurekaProcessInfo = ^TEurekaProcessInfo;
```

8.22 TEurekaProcessesList type

This is the TEurekaModuleList type:

Unit ExceptionLog.

```
TEurekaProcessesList = class(TList)
...
public
  ...
  property Items[Index: Integer]: PEurekaProcessInfo[74] read GetItem; default;
end;
```

8.23 TEurekaFunctionsList type

This is the TEurekaFunctionsList type:

Unit ExceptionLog.

```
  TEurekaFunctionsList = class(TList)
  ...
  public
    ...
    property Items[Index: Integer]: PEurekaFunctionInfo 75 read GetItem; default;
  end;
```

8.24 TEurekaFunctionInfo type

This is the TEurekaFunctionInfo type:

Unit ExceptionLog.

```
  TEurekaFunctionInfo = packed record
    Name: string;
    Addr: DWord;
    Size: DWord;
  end;
  PEurekaFunctionInfo = ^TEurekaFunctionInfo;
```

8.25 TEurekaModuleType type

This is the TEurekaModuleType type:

Unit ExceptionLog.

```
  TEurekaModuleType = (mtUnknown, mtMainModule, mtBorlandPackage, mtOSLibrary);
```

8.26 TEurekaExtraInformation type

This is the TEurekaExtraInformation type:

Unit ExceptionLog.

```
  TEurekaExtraInformation = packed record
    EurekaVersion:    Word;
    CompilationDate:  TDateTime;
    Options:          TEurekaModuleOptions 69;
    DebugInformation: TMemoryStream;
  end;
```

8.27 TEurekaAction type

This is the TEurekaActionType type:

Unit ExceptionLog.

```
  TEurekaActionType = (atShowingExceptionInfo, atShowedExceptionInfo,
atSavingLogFile,
                       atSavedLogFile, atSendingEmail, atSentEmail,
atSendingWebMessage,
                       atSentWebMessage, atTerminating);
```

8.28 TForeground type

This is the TForegroundType type:

Unit ECore.

```
  TForegroundType = (ftGeneral, ftCallStack, ftModules, ftProcesses, ftAssembler,
ftCPU);
```

8.29 TMessageType type

This is the TMessageType type:

Unit ETypes.

```
  TMessageType = (
    // General Messages...
    mtInformationMsgCaption, mtQuestionMsgCaption, mtErrorMsgCaption,
    // Exception Dialog (EurekaLog style)...
    mtDialog_Caption, mtDialog_ErrorMsgCaption,
    mtDialog_GeneralCaption, mtDialog_GeneralHeader,
    mtDialog_CallStackCaption, mtDialog_CallStackHeader,
    mtDialog_ModulesCaption, mtDialog_ModulesHeader,
    mtDialog_ProcessesCaption, mtDialog_ProcessesHeader,
    mtDialog_AsmCaption, mtDialog_AsmHeader,
    mtDialog_CPUCaption, mtDialog_CPUHeader,
    mtDialog_OKButtonCaption, mtDialog_TerminateButtonCaption,
    mtDialog_RestartButtonCaption, mtDialog_DetailsButtonCaption,
    mtDialog_CustomButtonCaption,  mtDialog_SendMessage,
    mtDialog_ScreenshotMessage, mtDialog_CopyMessage, mtDialog_SupportMessage,
    // Exception Dialog (MS style)...
    mtMSDialog_ErrorMsgCaption, mtMSDialog_RestartCaption,
    mtMSDialog_TerminateCaption, mtMSDialog_PleaseCaption,
    mtMSDialog_DescriptionCaption, mtMSDialog_SeeDetailsCaption,
    mtMSDialog_SeeClickCaption, mtMSDialog_HowToReproduceCaption,
    mtMSDialog_EmailCaption, mtMSDialog_SendButtonCaption,
    mtMSDialog_NoSendButtonCaption,
    // Log...
    mtLog_AppHeader, // Application...
    mtLog_AppStartDate, mtLog_AppName, mtLog_AppVersionNumber,
    mtLog_AppParameters, mtLog_AppCompilationDate, mtLog_AppUpTime,
    mtLog_ExcHeader, // Exception...
    mtLog_ExcDate, mtLog_ExcAddress, mtLog_ExcModuleName, mtLog_ExcModuleVersion,
    mtLog_ExcType, mtLog_ExcMessage, mtLog_ExcID, mtLog_ExcCount, mtLog_ExcStatus,
    mtLog_ExcNote,
    mtLog_UserHeader, // User...
    mtLog_UserID, mtLog_UserName, mtLog_UserEmail, mtLog_UserCompany,
    mtLog_UserPrivileges,
    mtLog_ActCtrlsHeader, // Active Controls...
    mtLog_ActCtrlsFormClass, mtLog_ActCtrlsFormText,
    mtLog_ActCtrlsControlClass, mtLog_ActCtrlsControlText,
    mtLog_CmpHeader, // Computer...
    mtLog_CmpName, mtLog_CmpTotalMemory, mtLog_CmpFreeMemory,
    mtLog_CmpTotalDisk, mtLog_CmpFreeDisk, mtLog_CmpSystemUpTime,
    mtLog_CmpProcessor, mtLog_CmpDisplayMode, mtLog_CmpDisplayDPI,
    mtLog_CmpVideoCard, mtLog_CmpPrinter,
    mtLog_OSHeader, // Operating System...
    mtLog_OSType, mtLog_OSBuildN, mtLog_OSUpdate,
    mtLog_OSLanguage, mtLog_OSCharset,
```

```
  mtLog_NetHeader, // Network...
  mtLog_NetIP, mtLog_NetSubmask, mtLog_NetGateway, mtLog_NetDNS1,
  mtLog_NetDNS2, mtLog_NetDHCP,
  mtLog_CustInfoHeader, // Custom Information...
  // Call Stack...
  mtCallStack_Address, mtCallStack_Name, mtCallStack_Unit,
  mtCallStack_Class, mtCallStack_Procedure, mtCallStack_Line,
  mtCallStack_MainThread, mtCallStack_ExceptionThread, mtCallStack_RunningThread,
  mtCallStack_CallingThread, mtCallStack_ThreadID, mtCallStack_ThreadPriority,
  mtCallStack_ThreadClass, mtCallStack_LeakCaption, mtCallStack_LeakData,
  mtCallStack_LeakType, mtCallStack_LeakSize, mtCallStack_LeakCount,
  // Send Dialog...
  mtSendDialog_Caption, mtSendDialog_Message, mtSendDialog_Resolving,
  mtSendDialog_Login, mtSendDialog_Connecting, mtSendDialog_Connected,
  mtSendDialog_Sending, mtSendDialog_Sent, mtSendDialog_SelectProject,
  mtSendDialog_Searching, mtSendDialog_Modifying, mtSendDialog_Disconnecting,
  mtSendDialog_Disconnected,
  // Reproduce Dialog...
  mtReproduceDialog_Caption, mtReproduceDialog_Request,
  mtReproduceDialog_OKButtonCaption,
  // Modules List...
  mtModules_Handle, mtModules_Name, mtModules_Description,
  mtModules_Version, mtModules_Size, mtModules_LastModified, mtModules_Path,
  // Processes List...
  mtProcesses_ID, mtProcesses_Name, mtProcesses_Description, mtProcesses_Version,
  mtProcesses_Memory, mtProcesses_Priority, mtProcesses_Threads, mtProcesses_Path,
  // CPU...
  mtCPU_Registers, mtCPU_Stack, mtCPU_MemoryDump,
  // Send Messages...
  mtSend_SuccessMsg, mtSend_FailureMsg, mtSend_BugClosedMsg, mtSend_UnknownErrorMsg,
  mtSend_InvalidLoginMsg, mtSend_InvalidSearchMsg, mtSend_InvalidSelectionMsg,
  mtSend_InvalidInsertMsg, mtSend_InvalidModifyMsg,
  // File Cracked Messages...
  mtFileCrackedMsg,
  // Exceptions...
  mtException_LeakMultiFree, mtException_LeakMemoryOverrun, mtException_AntiFreeze,
  // Others...
  mtInvalidEmailMsg);
```

8.30 TExceptionDialogOptions type

This is the TExceptionDialogOptions type:

Unit ECore.

```
TExceptionDialogOptions = set of TExceptionDialogOption 77;
```

8.31 TExceptionDialogOption type

This is the TExceptionDialogOption type:

Unit ECore.

```
TExceptionDialogOption = (edoSendErrorReportChecked, edoAttachScreenshotChecked,
                          edoShowCopyToClipOption, edoShowDetailsButton,
                          edoShowInDetailedMode, edoShowInTopMostMode,
                          edoUseEurekaLogLookAndFeel, edoShowSendErrorReportOption,
                          edoShowAttachScreenshotOption, edoShowCustomButton);
```

8.32 TCommonSendOptions type

This is the TCommonSendOptions type:

Unit ECore.

```
  TCommonSendOptions = set of TCommonSendOption 78;
```

8.33 TCommonSendOption type

This is the TCommonSendOption type:

Unit ECore.

```
  TCommonSendOption = (sndShowSendDialog, sndShowSuccessFailureMsg, sndSendEntireLog,
                       sndSendXMLLogCopy, sndSendScreenshot, sndUseOnlyActiveWindow,
                       sndSendLastHTMLPage, sndSendInSeparatedThread,
sndAddDateInFileName,
                       sndAddComputerNameInFileName);
```

8.34 TCallStackOptions type

This is the TCallStackOptions type:

Unit ECore.

```
  TCallStackOptions = set of TCallStackOption 78;
```

8.35 TCallStackOption type

This is the TCallStackOption type:

Unit ECore.

```
  TCallStackOption = (csoShowDLLs, csoShowBPLs, csoShowBorlandThreads,
                      csoShowWindowsThreads, csoDoNotStoreProcNames);
```

8.36 TBehaviourOptions type

This is the TBehaviourOptions type:

Unit ECore.

```
  TBehaviourOptions = set of TBehaviourOption 78;
```

8.37 TBehaviourOption type

This is the TBehaviourOption type:

Unit ECore.

```
  TBehaviourOption = (boPauseBorlandThreads, boDoNotPauseMainThread,
                      boPauseWindowsThreads, boUseMainModuleOptions,
boCopyLogInCaseOfError,
                      boSaveCompressedCopyInCaseOfError, boHandleSafeCallExceptions,
                      boCallRTLExceptionEvent, boCatchHandledExceptions);
```

8.38 TLogOptions type

This is the TLogOptions type:

Unit ECore.

```
TLogOptions = set of TLogOption 79;
```

8.39 TLogOption type

This is the TLogOption type:

Unit ECore.

```
TLogOption = (loNoDuplicateErrors, loAppendReproduceText,
              loDeleteLogAtVersionChange, loAddComputerNameInLogFileName,
              loSaveModulesAndProcessesSections, loSaveAssemblerAndCPUSections);
```

8.40 TShowOptions type

This is the TShowOptions type:

Unit ECore.

```
TShowOptions = set of TShowOption 79;
```

8.41 TShowOption type

This is the TShowOption type:

Unit ETypes.

```
TShowOption = (
  // Application
  soAppStartDate, soAppName, soAppVersionNumber, soAppParameters,
  soAppCompilationDate, soAppUpTime,
  // Exception
  soExcDate, soExcAddress, soExcModuleName, soExcModuleVersion, soExcType,
  soExcMessage, soExcID, soExcCount, soExcStatus, soExcNote,
  // User
  soUserID, soUserName, soUserEmail, soUserPrivileges, soUserCompany,
  // Active Controls
  soActCtlsFormClass, soActCtlsFormText, soActCtlsControlClass,
  soActCtlsControlText,
  // Computer
  soCmpName, soCmpTotalMemory, soCmpFreeMemory, soCmpTotalDisk,
  soCmpFreeDisk, soCmpSysUpTime, soCmpProcessor, soCmpDisplayMode,
  soCmpDisplayDPI, soCmpVideoCard, soCmpPrinter,
  // Operating System
  soOSType, soOSBuildN, soOSUpdate, soOSLanguage, soOSCharset,
  // Network
  soNetIP, soNetSubmask, soNetGateway, soNetDNS1, soNetDNS2, soNetDHCP,
  // Custom Data
  soCustomData);
```

8.42 TEmailSendMode type

This is the TEmailSendOptions type:

Unit ECore.

```
  TEmailSendMode = (esmNoSend, esmEmailClient, esmSMTPClient, esmSMTPServer);
```

8.43 TWebSendMode

This is the TEmailSendOptions type:

Unit ECore.

```
  TWebSendMode = (wsmNoSend, wsmHTTP, wsmHTTPS, wsmFTP, wsmBugZilla, wsmFogBugz,
wsmMantis);
```

8.44 EFrozenApplication type

This is the EFrozenApplication type:

Unit ExceptionLog.

```
  EFrozenApplication = class(Exception);
```

8.45 TTerminateBtnOperation

This is the TTerminateBtnOperation type:

Unit ECore.

```
  TTerminateBtnOperation = (tbNone, tbTerminate, tbRestart);
```

8.46 EEurekaLogGeneralError

This is the EEurekaLogGeneralError type:

Unit ExceptionLog.

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

8.47 TEurekaLogErrorCodes type

This is the TEurekaLogErrorCodes type:

Unit ExceptionLog.

```
  TEurekaLogErrorCode = (eeNone, eeShowError, eeLogError, eeEmailMAPIError,
                         eeEmailShellError, eeEmailSMTPError, eeWebHTTPError,
                         eeWebHTTPSError, eeWebFTPError, eeWebTrakerError);
```

Part

9 EurekaLog Viewer

9.1 How to use the Viewer

Use this Viewer to analyze and process all the bug reports received from EurekaLog (.elf files).

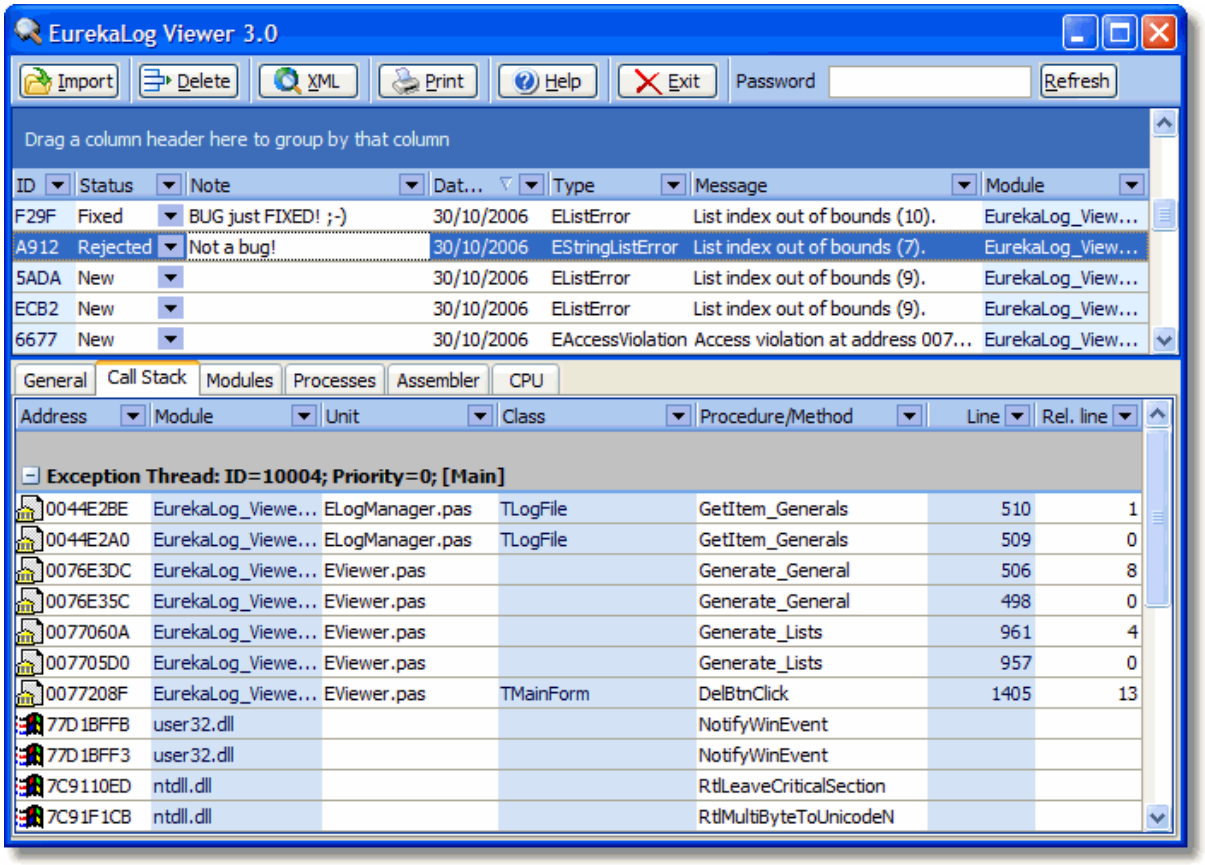
You can open this program with a double-click on the Log file (.elf) or using the "View Exception Log..." 11 new item in the Project IDE menu.

To analyze an encrypted Log file you must enter the password in the relative "Password" field pressing the "Refresh" button.

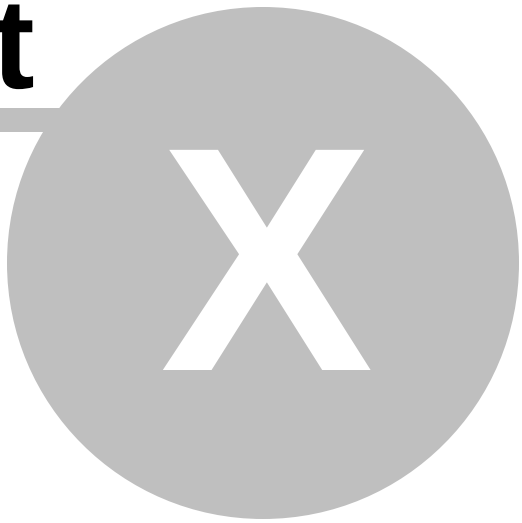
Use the "XML" button to export the Log file in XML format.

Use the Delete button to remove the currently selected bug report.

use the Print button to print the current selected bug report.



Part



10 Compatibility

10.1 Old 3.x version

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 36 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.

10.2 Changed from the old 4.x version

These are the changed from the old 4.x version to the new 4.5 version:

1. Changed the EmailObject property in EmailSubject 69
2. Changed the TEmailSendOptions 80 type to "(esoNoSend, esoEmailClient, esoSMTPClient, esoSMTPServer)"
3. Changed the TLogOption 79 type to "(loNoDuplicateErrors, loAppendReproduceText)"
4. Changed the AppendToLog property to AppendLogs 69
5. Changed the "MuteMode" property to "ShowExceptionDialog 69" (with inverted sense)
6. Changed the TForegroundType 76
7. Changed the TMessageType 76 type
8. Changed the TShowOption 79 type
9. EMailSendConfirm property removed
10. EResFile.pas removed

That's all.

10.3 Changed from the old 4.5.x version

These are the changed from the old 4.5.x version to the new 5.0 version:

1. SMTPShowDialog property removed (replaced by CommonSendOptions.sndShowSendDialog 78)
2. SendEntireLog property removed (replaced by CommonSendOptions.sndSendEntireLog 78)
3. EurekaLogLook property removed (replaced by ExceptionDialogOptions.edoUseEurekaLogStyle 77)
4. ShowExceptionDialog property removed (replaced by ExceptionDialogOptions.edoShowExceptionDialog 77)
5. EmailSendOptions property renamed in EmailSendMode 80 (with suffix replaced from 'eso' to 'esm')
6. TEurekaExtraInformation.CompiledDate replaced with TEurekaExtraInformation.CompilationDate 75
7. Added the 'so' prefix at every TShowOption 79 items

That's all.

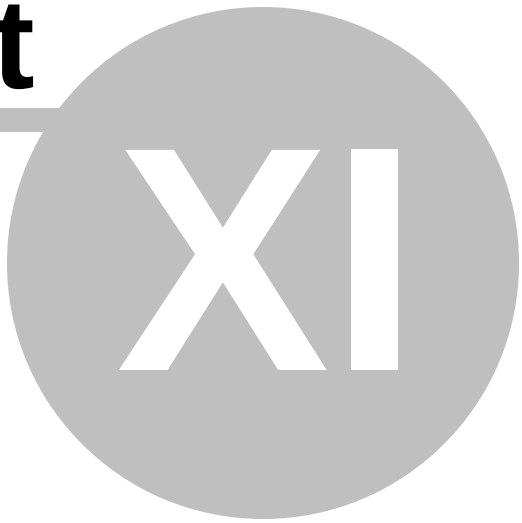
10.4 Changed from the old 5.x version

These are the changed from the old 5.x version to the new 6.0 version:

1. CustomDataRequest 42 parameter event changed ("CustomData: String" parameter is replaced with the new "DataFields: TStrings")
2. CustomFieldsRequest event removed (replaced with the new "CustomWebFieldsRequest 43" event)
3. FreezeMessage property removed (replaced with a "Messages Text Tab 18" field)
4. ExceptionClassName property removed (replaced with the new "ExceptionsFilters 69")
5. ExceptionMessage property removed (replaced with the new "ExceptionsFilters 69")
6. ShowTerminateBtn property removed (replaced with the "TerminateBtnOperation 69 = tbnone" value)
7. TExceptionDialogOption.edoShowExceptionDialog replaced with TExceptionDialogType 71
8. TExceptionDialogOption.edoSendEmailChecked replaced with TExceptionDialogType.edoSendErrorReportChecked 71
9. TCommonSendOption.sndCompressAllFiles removed
10. TBehaviourOption.boActivateCrashDetection replaced with AutoCrashOperation 69
11. TLogOption.loSaveModulesSection replaced with TLogOption.loSaveModulesAndProcessesSections 79
12. TLogOption.loSaveCPUSection replaced with TLogOption.loSaveAssemblerAndCPUSections 79

That's all.

Part



11 Enterprise version

11.1 Recompiling

Enterprise version is delivered with full source code.

If you want to recompile EurekaLog, you must close all open projects and then open the ExceptionExpertX.dpk (or CExceptionExpertX.dpk for C++Builder) package, where X is the Delphi/C++Builder version number, and then recompile the package.

Part

12 Command-line compiler

12.1 To use the command-line compiler

EurekaLog provides two command-line compilers: **ecc32.exe** for Delphi and **emake.exe** for C++ Builder. They are located in the same directory as the **dcc32.exe/make.exe** compilers.

These new compilers integrate the standard **dcc32.exe/make.exe** features plus the new 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.

Using the new command-line compiler, you can compile your projects just like when you use the standard compiler, adding to your projects the new EurekaLog features.

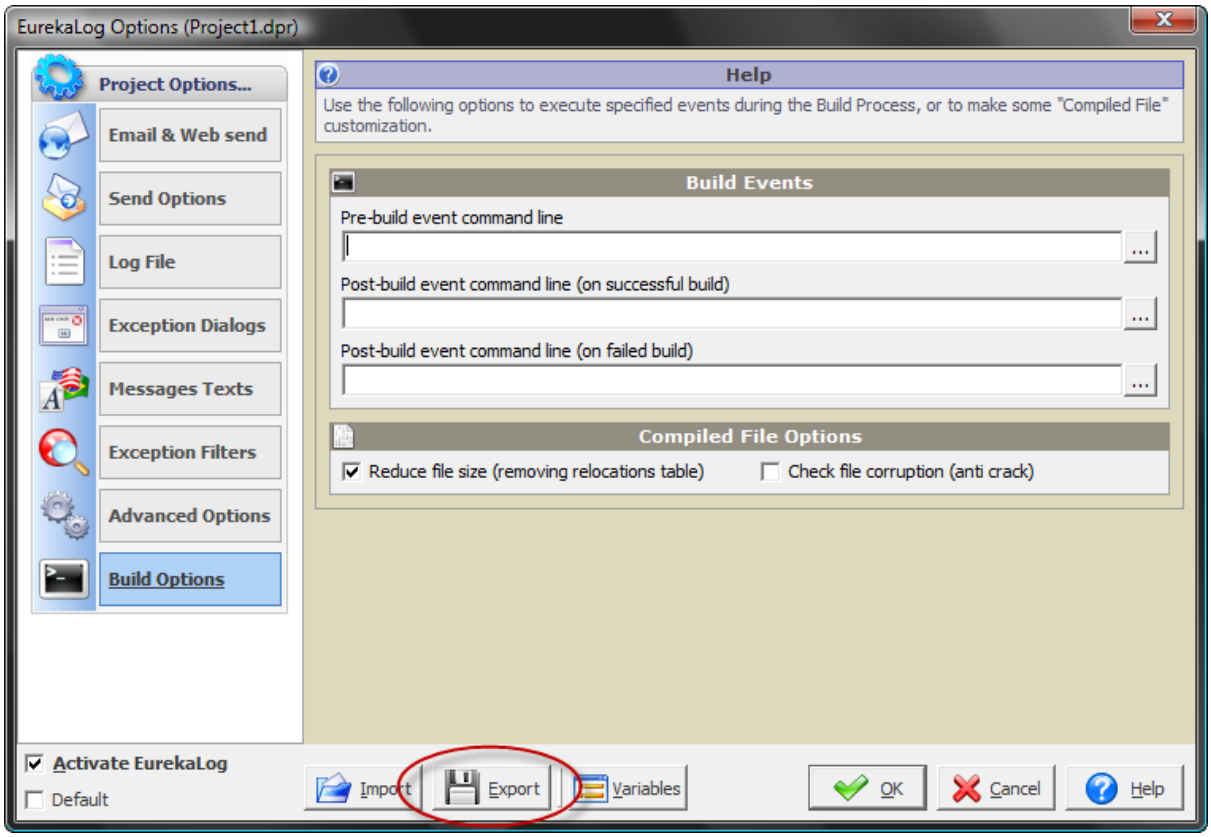
If it is used without any EurekaLog custom parameters (see below) it will simply call the standard compiler (dcc32.exe or make.exe) and after that it will alter the compiled file adding the EurekaLog options and debug data. The EurekaLog options are taken from the standard project options file:

DOF	Delphi 3/7
BDSPROJ	Delphi 2005/2006
DPROJ	Delphi 2007 and later
BPR	C++Builder 5/6
BDSPROJ	C++Builder 2006
CBPROJ	C++Builder 2007 and later

The EurekaLog command line compilers add the following new options:

--el_config

This is used to compile your project with a different EurekaLog option file. To create a EurekaLog option file you can use the **Export** button on the EurekaLog options dialog.



Example

--el_config"Project1.eof"

--el_alter_exe

This option instructs the EurekaLog compiler to not compile the project and only add the EurekaLog options and debug data into a compiled application (you can also optionally specify the compiled application's filename).

It is useful when used with 3rd party tools that are not able to directly run the ecc32.exe command-line compiler.

Delphi Example

--el_alter_exe"ProjectFile.dpr[;ProjectFile.exe]"

C++Builder Example

--el_alter_exe"MakeFile.mak"
--el_alter_exe"ProjectFile.bpr[;ProjectFile.exe]"

Don't use the standard command-line compiler anymore: use only the new EurekaLog command-line compiler.

Delphi command-line compiler

ecc32.exe "ProjectFile.dpr"

C++Builder command-line compiler

emake.exe -B -f"MakeFile.mak"

*Note: the **emake.exe** command-line compiler did not support ".bpr" files (it only supported ".mak" files). To generate the ".mak" file from a ".bpr" you can use the **bpr2mak.exe** command-line program.*

Part

13 Support

13.1 FAQ

Question: How does EurekaLog work?
Answer: EurekaLog perfectly integrates itself with Delphi's IDE and builds (with the normal compile command) your applications with the new capability to intercept every exception and trace a detailed and customizable log of every code point executed between the application's start and the point where an exception is raised (showing address, module, unit, class, method, line number and much more).

Question: How much does EurekaLog increase compiled file size?
Answer: EurekaLog increases the compiled file size by about 300 Kbyte plus about 0.5%.

Question: Can EurekaLog decrease an application's performance?
Answer: Definitely NOT, because it works only when an exception is raised. Abiliting the Memory Leaks trapping it can decrease your application performances of a max of 5%.

Question: What would cause the compiled file size to increase by more than decleared increasing? (Happens sometimes, but very rarely).
Answer: When you set the "Project/Options/Debugging/Use Debug DCUs" project option, Delphi includes all VCL source-code data in the EurekaLog debug information. This increases the final size of compiled file.

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.

Question: Does EurekaLog work with DLLs and BPLs?
Answer: Yes it does.

Question: What does EurekaLog do when it intercepts an error?
Answer: It displays a window to communicate the error in a graphic (GUI) application, it shows a text in a Console application and shows an HTML document in a Web application.

Question: What is the log file's extension?
Answer: It is *.ELF (EurekaLog File).

Question: Does EurekaLog intercept every type of exception-based error?
Answer: Yes it does. EurekaLog is able to intercept every type of exception-based error, Thread, Console and Internet Web exceptions included.

Question: How many errors can EurekaLog store in the file log?
Answer: There are no limits.

Question: When there are errors already verified and saved in the log file, is it possible to avoid further savings?
Answer: Yes it is. You must set the "Do not save duplicate errors" option in the "Exceptions Log Options..." menu.

13.2 On-line support

Feel free to visit http://www.eurekalog.com/support.php or send an e-mail to support@eurekalog.com

Part

14 Unsupported applications

EurekaLog support natively more applications types (see "features" 4 topic for further details), but not all exists applications types, so the scope of this little tutorial is explain how to can use EurekaLog in unsupported applications types.

Basically the steps to do are two:

- create a OnException event for the application type *(used to call EurekaLog manually)*;
- create a ExceptionNotify 36 event for EurekaLog *(used to customize the EurekaLog behavior)* .

NOTE: handling Web application you must disable the "Show Exception Dialog" 16 option to inhibit the Exception Dialog showing.

So a little example *(to manage unsupported Web modules)* can become as follow:

```
// Event uses to call manually EurekaLog...
procedure TWebModule1.WebModuleException(Sender: TObject; E: Exception; var Handled:
Boolean);
begin
   Handled := True;
   StandardEurekaNotify(E, ExceptAddr); // Call manually EurekaLog
end;

// Event used to send to Browser the EurekaLog error page...
procedure TWebModule1.EurekaLog1ExceptionNotify(EurekaExceptionRecord:
TEurekaExceptionRecord; var Handled: Boolean);
begin
   Response.Content := GenerateHTML(
     EurekaExceptionRecord.LogText, // Exception Log plain text
     True);                         // Add the JavaScript 'back' button at the page
bottom
end;
```

Part

15 Memory Leaks Limits

The EurekaLog Memory Leaks Detection has same limits derives from its internal structure.

EurekaLog detects the memory leaks at the program exit, so at this state the program has just freed all its non-static resources (resources allocated at run-time as Object created with the Create method, variables allocated with the GetMem function, or constants arrays).

This technique generating the following limitations:

1. currently this feature is available only in Delphi (not C++Builder);
2. the Leak report automatically hide the Assembler and CPU sections;
3. during the Leaks handling EurekaLog did not execute its events (this because all the needed memory resources are just freed);
4. a little performances decreasing (no more that about 5%);
5. a little memory usage increasing (about 300 bytes for every memory allocations);
6. EurekaLog can catches a max of 1024 leaks (to avoid the creation of a too BIG report file);
7. EurekaLog isn't able to catches memory leaks in projects compiled with Run-Time Packages.

SUGGESTION:
It's possible customize the EurekaLog options, related to the Leaks detection, before the program exit (using an unit "finalization" section, a TForm.OnDestroy event, or just putting it at the end of the project source code).

EXAMPLE:

```
program Project1;

uses
  ExceptionLog,
  ECore,  // Needed unit
  ETypes, // Needed unit
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

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

  // EurekaLog customization (for the Leaks Detection)...
  CurrentEurekaLogOptions.EMailSendMode  := esmSMTPServer;
  CurrentEurekaLogOptions.EMailSubject   := 'BUG Report';
  CurrentEurekaLogOptions.EMailMessage   := 'This is a BUG Report.';
  CurrentEurekaLogOptions.EMailAddresses := 'support@yoursite.com';
```

```
end.
```

Index

- A -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- W -