**I receive a lot of error reports from users of my software every day. Does crashrpt have a server-side component that would process crash reports automatically?**

You may try the [CrashFix Server](http://crashfixsoftware.com/) that can process crash reports sent by CrashRpt.

[http://crashrpt.googlecode.com/svn/wiki/img/CrashRpt/download_crashfix.png](http://crashfixsoftware.com/)

## Why is it strongly recommended that I compile crashrpt myself?

It is strongly recommended that you compile CrashRpt yourself to ensure CrashRpt uses the same C run-time DLLs as your application.

CrashRpt distribution already has compiled CrashRpt binaries, but it is not recommended to use them with your software, because your software may use different C run-time DLLs, and CrashRpt won't be able to intercept exceptions in your C run-time libraries.

## All my projects use multi-byte character set, so I changed it also in crashrpt library, but I experience some compilation problems. What do I do?

It is not recommended to modify character set related settings in CrashRpt solution projects. CrashRpt should always be compiled using the UNICODE character set. In your multi-byte application, you can use A-suffixed functions from CrashRpt API. For example, use crInstallA(),crAddFile2A() and so on. Typically, you use a character-set-independent mapping of a function name, for example crInstall() name automatically expands into crInstallA() in a multi-byte program, and into crInstallW() in a wide-char program.

**I install crashrpt to my MFC app, but exceptions are not caught. What do I do?**

You can override your CWinApp::Run() method and install CrashRpt there. In the CPP file containing your App class, add the following code:

int CYourApp::Run()   
{  
  // Call your crInstall code here ...  
  
  BOOL bRun;  
  BOOL bExit=FALSE;  
  while(!bExit)  
  {  
    bRun= CWinApp::Run();  
    bExit=TRUE;  
  }  
  return bRun;  
}

## How do I use crashrpt in a multi-threaded application?

You should use CrashRpt API functions to set exception handlers in the beginning of thread procedure of each of your worker threads. Do not forget to unset handlers in just before exiting the thread procedure.

## I want to uniquely identify the installation by what the error reports are sent. What do I do?

In CrashRpt v.1.2.1, we introduced a function crAddProperty() specially for purposes of extending the crash description XML as you wish. You can create the installation GUID in your program, store it in the registry or in a file and pass this GUID to the crAddProperty() function, so the named property will be added to the crashrpt.xml and you will be able to identify the installation and associate the bugs together.

The following code example shows how to do this (code provided by tdev):

  // ... upon program start, check if GUID is not existing, if not create one:  
  GUID \*g = new GUID();  
  CoCreateGuid(g);  
    
  char buf[120];  
  sprintf( buf, "%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x",  
    g->Data1, g->Data2, g->Data3, UINT(g->Data4[0]), UINT(g->Data4[1]), UINT(g->Data4[2]),   
    UINT(g->Data4[3]), UINT(g->Data4[4]), UINT(g->Data4[5]), UINT(g->Data4[6]), UINT(g->Data4[7]));  
  delete g;  
  std::string guid = std::string(buf);  
  
  //... later, add it to the crash report:  
  crAddProperty("SystemGUID", guid);

## My software's worker thread hangs up on users machine (but the main thread is alive and no crash happens). How do I debug the problem?

To collect information about such error, you may provide an ability to generate error report manually, for example on a key combination press. When the key combination is pressed, generate error report manually using crGenerateErrorReport() function.

## Does crashrpt support a non-GUI (silent) mode?

Yes, CrashRpt supports non-GUI mode. In the non-GUI mode, CrashRpt doesn't show any windows and sends the report without any user interaction. You can use such mode if your application is a service which does not interact with user. However, it is not recommended to use the non-GUI mode with regular GUI applications that create windows and interact with user.

## Does crashrpt support automatic restart of the application that crashes?

Yes, CrashRpt supports automatic application restart. But the restart is performed if two conditions are satisfied:

* If at least 60 seconds have elapsed since the application start. This is done to avoid cyclic restarts in case when crash occurs on application start up.
* If the user has provided his consent to restart the app.

## What is the recommended way of CRT linkage for crashrpt to work properly?

It is recommended that you configure your project to use C runtime libraries (CRT) as Multi-threaded DLL (/MD) for Release configuration. This is the way recommended in MSDN.

In the Solution Explorer window, right-click your project and open project Properties. Then choose Configuration Properties->C/C++->Code Generation. In the Runtime Library field, choose Multi-threaded DLL (/MD).

This should be done for all projects in your solution to ensure all modules will share single CRT DLL.

## Do program optimizations (/O compiler switch) affect the minidump generation?

Yes, if you enable optimizations, the stack trace recovered from minidump may be incomplete.

We recommend turning off Frame Pointer Omission (FPO) optimization in release version configurations, as this optimization does not really give an appreciable gain, but greatly complicates the analysis of the dump: the /Oy compiler option makes using the debugger more difficult because the compiler suppresses frame pointer information. Moreover, in Visual Studio 2010 this optimization is disabled by default.

If you can't afford yourself to turn the optimization off, you can manage some error log file which is not affected by optimizations.

## What is the minidump type? What minidump type should I use?

The minidump type defines what information is contained in the minidump. The default minidump type is MiniDumpNormal. However, theMiniDumpNormal contains only essential info required to recover the stack traces for each thread. To recover the state of global variables, other minidump types should be used. For additional info, see the [Effective Minidumps](http://www.debuginfo.com/articles/effminidumps.html) article.

**What crashrpt files do I distribute with my software release package?**

It is recommended that you distribute the following files with your software release (XXXX is the version number):

* CrashRptXXXX.dll
* CrashSenderXXXX.exe
* dbghelp.dll
* crashrpt\_lang.ini

## What is crash GUID?

A crash description XML contains CrashGUID tag. The crash GUID is an unique identifier that is assigned to the error report. GUIDs are generated using special algorithm which guarantees very low probability of generating the same GUID twice, even on different user machines. The crash GUID can be used, for example, as the primary key if you plan to store crash reports in a database.

## What is CrashRptProbe API?

The CrashRptProbe API is the application programming interface for processing error reports generated by crashrpt. The API includes several functions you can use in your own error report processing tool. The API is used internally by the crprober.exe tool.

The error report processing functionality is encapsulated inside of CrashRptProbe.dll. Internally CrashRptProbe.dll uses functions provided bydbghelp.dll (Microsoft Debug Help Library) for loading the minidump file contained in the report.

## How does crashrpt work internally?

CrashRpt consists of two core modules: CrashRpt.dll and CrashSender.exe.CrashRpt.dll contains functionality for intercepting exceptions in a client software. CrashSender.exe contains functionality for compressing and sending error reports to the software's support team.

It is unsafe to do complex operations in the crashed program which may be unstable. CrashRpt is separated into these modules to be able to close the application which have crashed and to continue sending the error report in CrashSender.exe in background.

Typically a process (executable file) loads CrashRpt.dll into its address space when process starts. This single instance of CrashRpt is shared between all DLL modules that are loaded into the address space of the process. Client application uses CrashRpt API functions to set exception handlers in the process once, typically in its main() function. In a multi-threaded program, client additionally sets exception handlers in the beginning of thread procedure of each worker thread.

## How can I use crashrpt without CrashSender.exe?

You can not generate crash reports without CrashSender.exe. It is unsafe to do complex operations in the crashed program which may be unstable. CrashSender.exe collects information about the crash, closes the application which have crashed and continues sending the error report in background.

## My software is not an EXE, but DLL (e.g. a plug-in for another software). Can I use crashrpt in such a case?

Generally, it is not recommended to use CrashRpt if your software is a plug-in DLL. It would be better if the client process (EXE) take care of exception handling instead of your DLL.

However, if you strongly wish, you may try to install exception handlers in your DllMain() function, but typically you can't guarantee that the client process (EXE) won't install its own exception handlers above yours. And if the client process uses several worker threads that call functions from your DLL, you would have to install per-thread exception handlers, too.

## Does using crashrpt affect performance of my application?

In most cases no, using CrashRpt doesn't affect performance of your application. CrashRpt doesn't perform any tasks in background and doesn't use additional memory. CrashRpt code is executed just when an exception is raised.

However, CrashRpt may affect performance of end user's computer if you run its desktop video capture feature (crAddVideo() function). In such a case, CrashRpt consumes some CPU resources and disk space to record video.

## Does crashrpt intercept *exactly* all exceptions that may occur in my program?

CrashRpt intercepts all exceptions that are allowed to intercept by C runtime libraries (CRT).

Note that some exceptions related to security can't be intercepted. Since CRT 8.0 you can't intercept security errors in your code. When buffer overrun is detected, CRT invokes Dr.Watson directly instead of calling unhandled exception filter. This is done because of security reasons and Microsoft doesn't plan to change this behavior.

For more information, see this [link](https://connect.microsoft.com/VisualStudio/feedback/ViewFeedback.aspx?FeedbackID=101337).

## I cannot seem to catch the crash at all, the application just closes. No crash window courtesy of Windows, nothing at all. It just poof, closes. What do I do?

The problem with silently disappearing application window may be related to [stack overflow](http://en.wikipedia.org/wiki/Stack_overflow). When stack is overflown, the process doesn't have enough memory even for invoking crash handler. So it looks like a silent exit.

The advice for you is to maintain a log file for your application. See this link for example of a [logging functionality](http://crashrpt.sourceforge.net/docs/html/simple_example.html).

The log file will help you to see where is the stack overflow happens. For example, if there is an [infinite recursion](http://en.wikipedia.org/wiki/Infinite_loop) in your code, you will see many log messages looking very similar.

## If I call C++ delete operator on an already deleted pointer, crashrpt cannot handle this situation correctly, most of the times it just stops working. Is this a bug in crashrpt?

No, this is not a bug in CrashRpt. For example, assume the following code:

CPoint\* pt = new CPoint;  
  
delete pt;  
delete pt;

Executing the code above results in heap corruption. What happens next is called "undefined behavior". Your program may crash immediatelly, crash randomly in some period of time or just continue working with corrupted memory. Obviously, CrashRpt cannot work correctly in this situation.

The recommended way of deleting a pointer is to delete it and set it to NULL. For example, consider making a macro for this:

#define SAFE\_DELETE(ptr) { delete(ptr); (ptr)=NULL; }

For more information, see this [post](http://stackoverflow.com/questions/2746677/what-happens-when-you-deallocate-a-pointer-twice-or-more-in-c).

## In some of received crash minidumps I see that the crash happened inCrashRpt.dll itself (mostly when it invoked InvalidParameterHandler). Is this normal?

Yes, this is normal. When a crash occurs, CrashRpt.dll checks if exception pointers structure presents. This structure is ususally allocated by the OS automatically when a SEH exception occurs (e.g., access violation reading NULL). If the structure doesn't present (which is true in case of invalid parameter exception), CrashRpt allocates the structure itself by getting the current state of the CPU registers. In such a case, you will see in the minidump that the crash originates inside of the CrashRpt.dll, but this is just an illusion. You should skip the stack frames that belong to CrashRpt.dll and look at the stack frames that belong to the modules of your application.