GSupport Introduction

General Support Classes

Version 1

Notice of Copyright

Bitspot AG
Rietlistrasse 1
CH 6345 Neuheim
Swizerland
www.bitspot.com
support@bitspot.com.

Phone: +41 41 755 11 22
Fax: +41 41 755 11 31

Table of Contents

1.General

This document explains the features of Bitspot AG General Support Library (GSupport.dll). It is aimed to the management personal and therefore is designed to grant a general view of the library's main features.

GSupport stands for "General (or Global) Support". As the name indicates, it is a library that exports classes for general support purposes.

The motivation leading for the developing of this product was the need of a good tracing tool and methods in order to detect (and sometime recover) program's failures. GSupport library covers these needs as follows:

- Trace Class for application messages
  - ⇒ Reporting to different outputs (file, socket, event viewer etc.)
  - ⇒ Different message level (debug, information, warning, error etc.)
- Exception classes
  - ⇒ Base class for common exception handling
  - ⇒ Exception class that convert system exception (SEH) into C++ exception and generates a "Dr. Watson" like crash file.
  - ⇒ Pre-defined derived classes (e.g. MFC & STL exception)
- Mole classes for extern access and manipulation of a trace object
  - ⇒ Via HTTP (IE, Netscape or Mozilla) protected by SSL
  - ⇒ Via Telnet
  - ⇒ Virtual implementation design for specific needs
- Critical Section class designed to assure correct use of the critical section object

These are the main classes of GSupport. This paper will concentrate mainly on these classes, the other export classes available in the library will be then shortly presented in one chapter.

Another special feature of GSupport is overwriting the application output/behavior setting with a command line parameter and an "INI" file. This means that the application shouldn't be recompiled with other setting in order to use other setting (e.g. changing the behavior when exception is caught).

For suggestions and/or questions please contact: support@bitspot.com

2.Non functional requirements

Software:

- GSupport was developed and tested using Microsoft Visual C++ version 7.1. It uses WIN32 API but does not use MFC, hence it should be possible to use other compilers. Tests, however, where not done.

Target platform:

- PC, Pentium 550MHz or higher with Microsoft Window XP or higher and Microsoft Visual C++ 7.1 or higher.

Note: Upon request (e.g. To the above mentioned Email) a version for windows operating system and VC++ compiler could be built.

3.The Trace Class

The trace class, CBS_Trace, is a class that provides a comfortable interface for tracing a chronological occurrences. It formats the row message with the date, time, calling module and the message type into one message

The following message types are supported:

- Debug (or Verbose) information
- Information
- Warning
- Error
- Fatal Error
- User Defined Exceptions
- MFC Exceptions
- STL Exceptions
- System Exceptions

The application could select none, one or more of the following outputs:

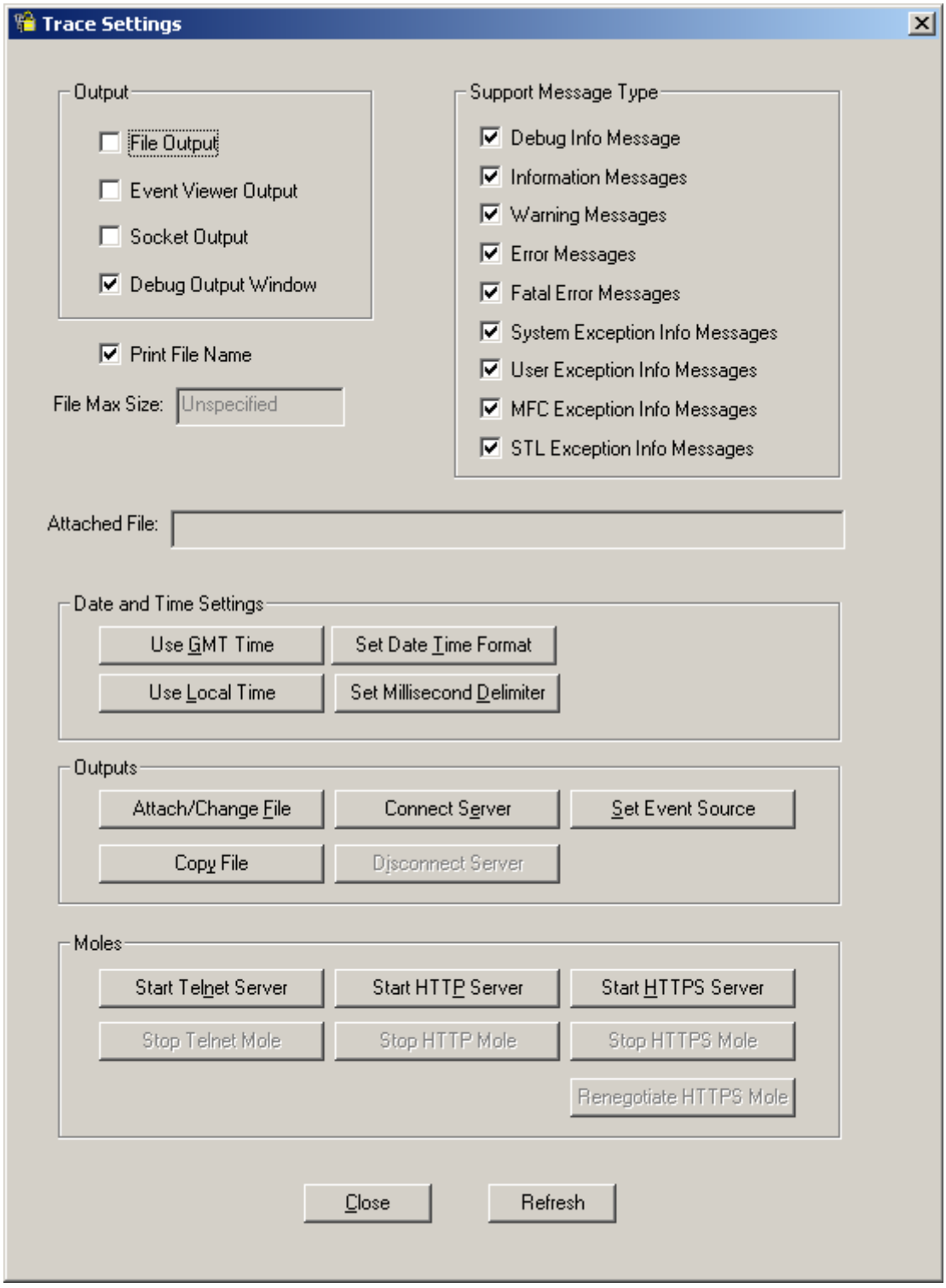
- File
  ⇒ Thread safe
  ⇒ open for exclusive write but shared read
  ⇒ copy/change is easy
- Socket
  ⇒ write output directly to desired server.
  ⇒ Recovery processing possible if the connection was broken.
- Console
- Event Viewer
- Debug Window (only if a debugger is present version)

The following output was generated from the row message “This is a sample warning!!!”.

```
10.05.03 13:19:38.330   Module: BS_Trace Sample Application Type: Warning Report
This is a sample warning!!!
```

The application could change the default format of the date and the time and select between UTC (GMT) or local time. The time output includes the millisecond so that tracking event occurrence order is more efficient.

Another important feature is the trace control dialog. A dialog box that the application can invoked during its execution, in order to provide an interface to the trace. At run time users can modify the current settings, manipulate the outputs and activate the moles, so that remote user (e.g. programmers) can access the library (see also moles description).



4.The Exception Classes

This class was actually developed in order to catch system exception to avoid application crash if a GPF (General Protection Fault e.g. division by zero or access violation) occurs. The goal was to register the failure and to be able to recover.

Since that throwing exception is a common C++ features, the system exception class is derived from a base class from which all other exception classes will be derived. This gives us the advantage of catching all exceptions in one block and handling them all the same way.

The application could decide how the library should react when an exception is caught by selecting none, one or more of the following flags:

- Write an error message to the associated trace object
- Display a (timed) message box
- Exit the thread if the exception is “non continuable” (non continuable exception are very rare)
- Exit the process if the exception is “non continuable”
- Exit the thread always
- Exit the process always

The application could set a callback functions that will be invoked to exit the process respective thread so that it could control the shutdown processing (free resources etc.).

These setting could be overwritten by an “INI” file if the application started with a certain command line parameter. The class is designed to open and fetch the settings from the file each time an exception is caught, so the user could change the setting during the runtime (without recompiling!!!).

For example starting the application with the command:

```
-BSEX-f MySettingFile.ini
```

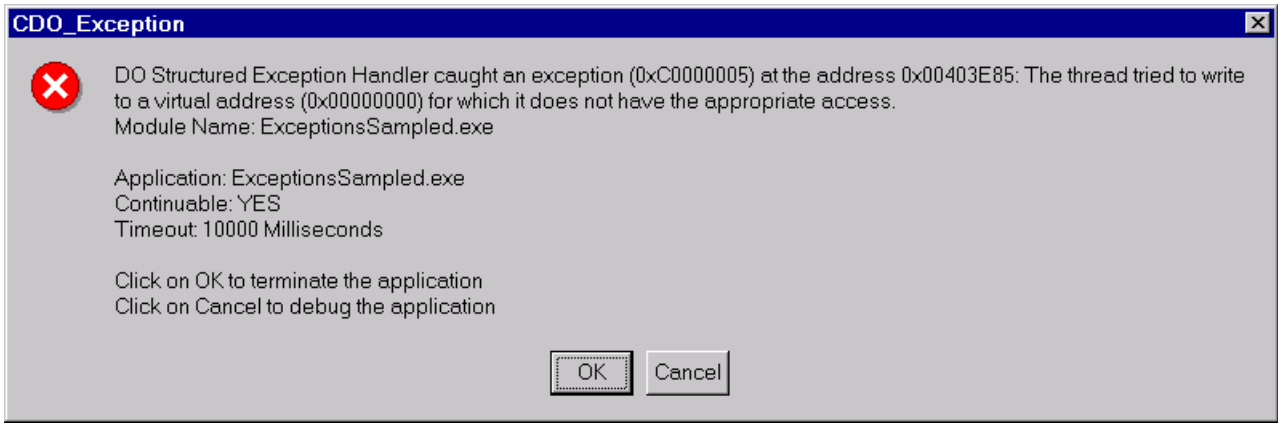
will cause the class to fetch the setting from MySettingFile.ini and to overwrite the setting specified by the application.

Sample for a setting file:

```
######################
[BS Exception Info]
Setting = 3
Timeout = 20000
TraceFile = TraceFromIni.txt
######################
```

Setting := a bit combination of the setting flags.
Timeout := the message box's timeout in milliseconds, -1 = infinite.
TraceFile := the trace file into which the exception should be written.

An example for a timed message box displaying a caught system exception triggered by NULL pointer.



Except from providing the application the possibility to recover after a GPF, the system exception class, CBS_SE_Exception, has an additional major feature: it generates a “Dr. Watson” like crash file into which the following information is logged:

- Task list
- Modules list
- The values saved in the Registers when the crash occurred
- Disassembly of the crash section
- Stack trace
- Stack row content
- Symbol table of all modules

In order to make the crash analysis easier, the symbol information, if available, will be saved in the file as well.

Of course the application control on these flags and enable/disable the one or the other.

Other exception classes implemented by GSupport are:

- CBS_SocketException to handle socket exceptions
- CBS_MFC_Exception, convert MFC exception into CBS_Exception
- CBS_STL_Exception, convert STL exception into CBS_Exception

Beside using these classes, an application could derive its own class from the CBS_Exception and to enjoy from all above mentioned features.

5.The Mole Classes

The mole classes are designed to provide a remote access to the running application. The main goals of these classes are:

- To retrieve information from the running application
  ⇒ View the content of the trace file
  ⇒ View the content of the crash file
  ⇒ View the current trace setting (e.g. report level)
  ⇒ View the current setting of the crash report
- To be able to change some settings
  ⇒ Change the current trace setting (e.g. report level)
  ⇒ Change the setting of the crash report
- To send commands to the application (e.g. restart)
  ⇒ Restart the application (e.g. with a batch)
  ⇒ Shutdown the application

This functionality is implemented in GSupport. The Moles, however, are designed as a base classes with many virtual functions, so that applications could derive its own Mole class and override some functions according to its specific needs. This construction gives the application the advantage to enjoy GSupport features and to add some features like:

- Add a challenge response mechanism for secure login
- Retrieve/Change value of application variables
- Retrieve/Change contents of configuration files
- Start batch files

There is actually no limit to what the application can do with these moles…

GSupport implements three moles: HTTP, HTTPS and Telnet.

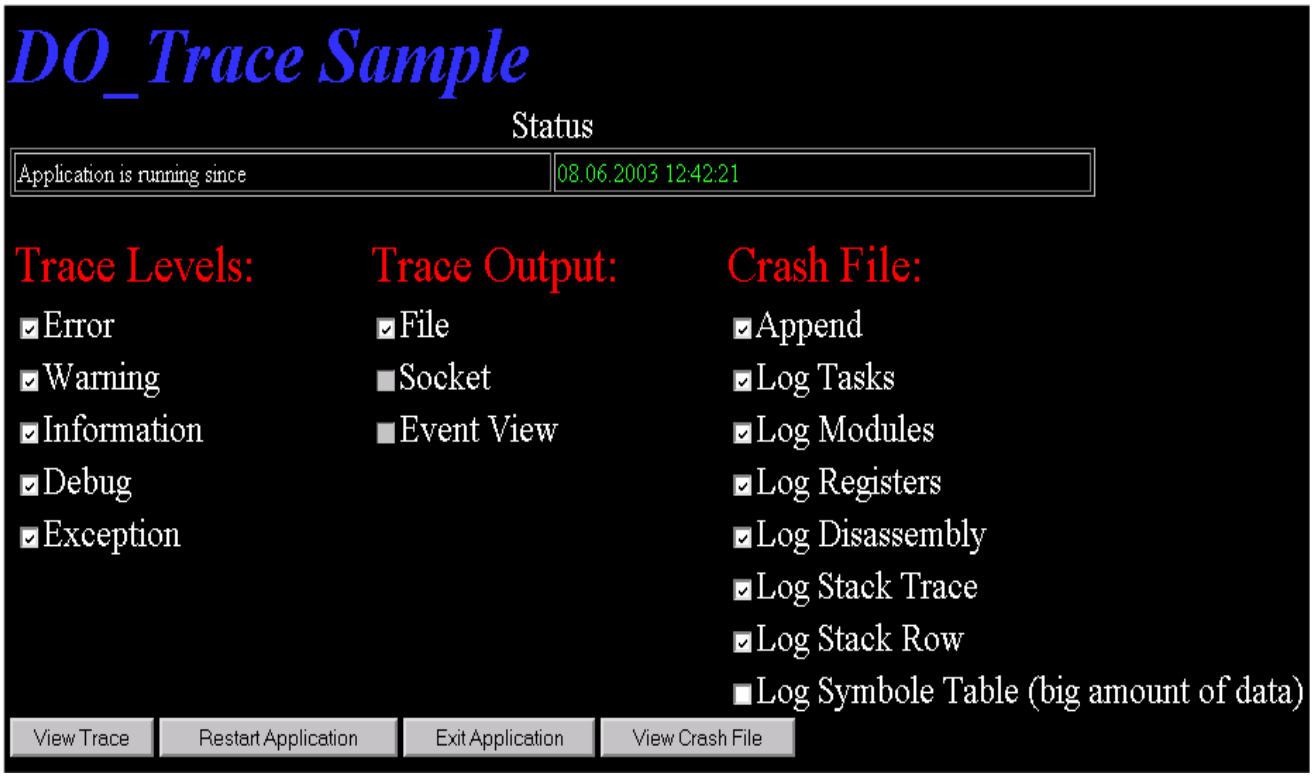
The HTTP and HTTPS Mole:

The HTTP Mole access the application from internet browser and provide a simple, comfortable user interface.

The HTTPS (RFC 2818) Mole is an extension that provide SSL/TLS protection of the connection.

The HTTPS mole allows only to browser that owns certificates (and their corresponding private keys) issued by trusted Certification Authorities (CA) to access the application. The application start the mole with a list of its trusted CA.

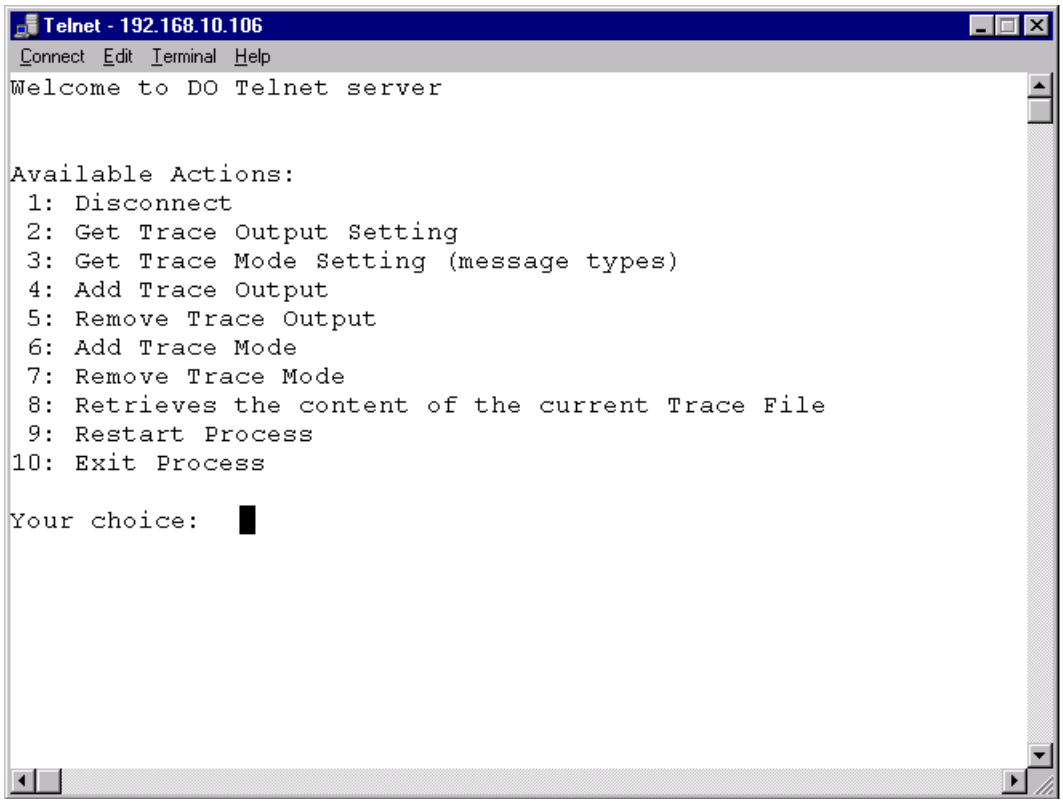
The following screenshot was taken from the browser connected with HTTP Mole:



The HTTP respective HTTPS connection was tested with IE 6.0, Netscape 6.2 and Mozilla 1.0.

The Telnet Mole:

The telnet mole is designed to run with a telnet client. The following screenshot presents a sample telnet console:



It is not as convenient as the HTTP though, certainly, good enough for simple commands.

6.The Critical Section Class

Many applications are designed to run in multithreading environment. Synchronization is a major issue in multithreading applications. Bad design might cause the application to hang without a warning or any other notification. Those kind of failures are hard to detect, specially if the cause is an inconsiderable sporadic race condition problems.

The goal of this class is to alert the user incase of a synchronization (of a critical section object) failure (e.g. deadlock or lockout). This goal is achieved by monitoring the synchronization objects. If there is suspicion for a bad synchronization, the user will be alert when the application shutdowns at the latest.

This class assures that:

- Every ‘Enter’ has a corresponding ‘Leave’
- A critical section object will not be deleted if a thread waits for it.
- Only the thread that owns the critical section may leave it.

The failure will be registered in a trace object.

Sample for possible trace message:

```
11.05.03 16:56:33.568   Module: CBS_CriticalSection   Type: Fatal Error Report
The Critical Section allocated in
D:\ExceptionsSample.cpp(192): was destroyed while 4 threads are waiting for it
to be freed!!!

11.05.03 16:53:32.818   Module: CBS_CriticalSection   Type: Fatal Error Report
The Critical Section allocated in
D:\ExceptionsSample.cpp(192): was destroyed while the thread 0x0000013E which
owns it (entered 4 times) did not leave it!!!


12.05.03 14:16:47.877   Module: CBS_CriticalSection   Type: Fatal Error Report
The Leave Critical Section call is BAD!!!
D:\ExceptionsSample.cpp(79): The thread 0x000000A2 does not owns the critical
section and therefore can not leave it!!!
```

For the convenience of the developers the location from which the bad call is done, is also printed to the output. The file name and line number are formatted in a way that double click on it in the debug window of VC++ compiler will open the file and set the cursor in the location from which the bad call is done.

The ‘Enter’ and ‘Leave’ are registered in a resources track object of GSupport (which will not be discussed in detail in this document). When the application exits, the resources track must be empty, if not it will print the entries that are still saved in it.

Sample for possible resources track message notifying a bad critical section handling:

```
**************************************
Bitspot's Resources Track found failure in the resources allocation!!! Date:
Sonntag, 11. Mai 2003 16:43:34.969
**************************************



//////////////////////////////////////
Initializing Critical Section at:
D:\ExceptionsSample.cpp(191): Initializing a new Critical Section

Entering (7) Critical Section
D:\ExceptionsSample.cpp(192): Main Thread!!!
D:\ExceptionsSample.cpp(219): Main Thread!!!
D:\ExceptionsSample.cpp(220): Main Thread!!!
D:\ExceptionsSample.cpp(221): Main Thread!!!
D:\ExceptionsSample.cpp(222): Main Thread!!!
D:\ExceptionsSample.cpp(228): Main Thread!!!
D:\ExceptionsSample.cpp(230): (null)

Leaving (6) Critical Section
D:\ExceptionsSample.cpp(217): Main Thread!!!
D:\ExceptionsSample.cpp(223): Main Thread!!!
D:\ExceptionsSample.cpp(224): Main Thread!!!
D:\ExceptionsSample.cpp(225): Main Thread!!!
D:\ExceptionsSample.cpp(226): Main Thread!!!
D:\ExceptionsSample.cpp(229): Main Thread!!!



**************************************
End of Bitspot's Resources Track Report.
**************************************
```

Note: The comments `"Main Thread!!!"` is an optional application defined comments.

Also here the file name and line number from which the calls are done are printed out for the convenience of the developers.

7.Other Classes

Apart from the classes discussed in the previous chapters, GSupport uses other helpful classes for it's own need. These classes are also exportable and therefore could be used by the application.

- Resource track class for assuring correct allocation/deallocation (including reference count) of resources.
- Timed message box class, a message box that will be closed after the given timeout elapsed.
- A class that stores the command line parameter in an argument vector (argv) which is more convenient to parse then character string.

8.Conclusion

GSupport is a strong library that provide a comfortable, reliable, fast methods to debug an application and track errors once occurred. It may save a lot of time in finding the causes for bugs (e.g. critical section deadlock) and provide recovery processing (e.g. exception). Apart from these features, it allows remote access to applications - a feature which makes supporting much more efficient and comfortable.

It is also a great support for developers when failure are caught and reported during the programming time (e.g. locating bad critical section handling).

Based on the above mentioned report it is obvious that GSupport is a real asset for support teams as well as developers.