Runtime errors when using external controllers

Contents

1 OVERVIEW

Bladed makes an effort to identify and report any exceptions that it encounters while running. In general, the later the version of Bladed, the better the identification of external controller runtime errors. However, certain categories of error are difficult to identify or intercept reliably, and this can lead to what Bladed identifies as "Unexpected terminations". Symptoms of this can include:

- $TE file reports an "Unexpected termination",
- The Bladed CMD window contains a Fortran error and traceback,
- The Bladed CMD window displays an error message and asks the user to press enter,
- A Windows popup appears with an error in it (this is unusual).

Due to the nature of the interface between Bladed and the external controller, many of the problems encountered cannot be caught reliably. This document describes some of the more common problems leading to "unexpected terminations", and where possible how to identify, debug and solve the problem.

2 POSSIBLE CAUSES

The interactions between the Bladed executable and the external controller dll are handled by the Windows operating system. Windows allows the executable *some* control over the running of the dll, and *some* feedback from the operation – but reserves the right to disregard these under certain circumstances. Because of this although there is error handling and reporting, it is not always able to report every eventuality.

The most common causes of ”unexpected terminations” are:

1. Wrong calling convention (__cdecl or __stcall).
2. Exceeding the bounds of an array – such as field [-1] or [1000] of the swap array (or any other internal array bounds error).
3. Using an incorrect function signature for DISCON (legacy) controllers – for instance, taking an array of doubles instead of an array of floats.
4. Throwing a C++ exception (including inbuilt ones, such as one thrown by vector or map), which the C interface cannot handle.
5. Out of memory error (for example: caused by an infinite loop).
6. A serious floating point error. Older versions of Bladed will attempt to continue operation, but later versions will stop when one is encountered.
7. Mixing the versions of dtbladed.exe and the ExternalControllerApi.dll, both of which are found in the Bladed installation directory.
8. Building the external controller with a non-compatible version of ExternalControllerApi.lib.
9. A bad operating system call or dependency in the external controller, such as using a third party dll that is missing.
10. Incompatible C++ runtimes in Bladed and the external controller.
11. Using a controller compiled in Debug on a non-development PC.

3 DETAILED DESCRIPTION OF CAUSES

3.1 Memory Access Violations

Memory access violations are caused by the program attempting to read or write memory which it has no right to. It is not possible to continue operation after an access violation is encountered, as all memory within the application becomes suspect, and to continue would risk either errors, or worse still: corrupted results.

Bladed makes an effort to identify and report when memory corruption has occurred, but it is not possible to identify the source. When Bladed encounters an access violation whilst running the external controller, it will print the following to the CMD window:

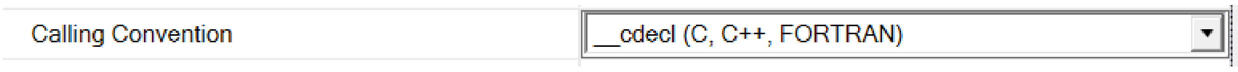
```
Bladed has experienced an unrecoverable memory violation fault coming from the external
controller and has had to terminate.  This may be caused by:
  - A segmentation fault in the external controller, or
  - The use of an external controller that accepts an
    incompatible set of arguments, or
  - Mixing calling conventions (cdecl and stdcall).
Press ENTER to close this window...
```

There can be many different sources of memory access violations. Those most often encountered by Bladed Support are:

3.1.1 Wrong calling convention

The calling convention (__cdecl or __stdcall) primarily concerns whose duty it is to delete the copies of an argument following a function call – the caller or the called. Different languages have different defaults, and Bladed defaults to the C/C++ convention. When the calling conventions don't match, either the arguments won't be deleted (causing a memory leak) or will be deleted twice – which causes an access violation.

This is a relatively common mistake, and very easy to fix in the external controller definition screen:



Calling Convention   __cdecl (C, C++, FORTRAN)

3.1.2 Exceeding the bounds of an array

In C, C++ and Fortran, if standard arrays are used, then it is possible to exceed the bounds of these arrays, reading or writing to an effectively random memory location.

The most common cause of this would be mistakes when accessing the swap array when using legacy controllers. Although directly specifying a field that is negative or very large would be quite unlikely, some of the fields reference the location of *other* fields in the swap array, and it can be quite easy to make mistakes (especially as Fortran arrays start at 1, and C/C++ arrays start at 0).

Identifying the source of these problems is extremely difficult. Often the most effective way is to debug through the external controller, and instructions for this process can be found in the *Bladed External Controller User Manual* (version 4.6 or later), in Appendix A.

3.1.3 Incorrect function signature for legacy external controllers

Bladed has a hard-coded expectation of what arguments the external controller takes. Should the external controller actually take a different set of arguments, this won't be checked during compilation, and the first error will be at runtime. Depending on the exact problem, this may or may not be trapped.

This problem may also occur when a legacy controller is called "CONTROLLER" – making Bladed assume that it is a functional-style controller – or vice versa. It is very important that legacy-style controllers (i.e. ones that use the swap array) are named "DISCON", and newer controllers are called "CONTROLLER" or "CONTROLLER_INIT". Some versions of Bladed allow the user to change the name of the external controller – but this should never be changed to "DISCON".

3.1.4 Throwing a C++ exception

The interface to the external controller is limited to the C level, to ensure compatibility with Fortran. Because of this, it cannot handle the C++ exception framework, and treats any errors raised as a memory exception.

3.2 Other Memory Exceptions

3.2.1 Out of memory error

With modern PCs, it would be very unusual to exceed the available memory without a serious programming error. The most likely source would be an infinite loop, which would eventually exceed the available memory if even a small amount of memory was allocated on each step.

Windows is relatively good at reporting out of memory errors, so you are likely to get an error from the operating system if this is the source of the unexpected termination.

3.3 Floating Point Errors

Floating point errors covers anything from a "floating point underflow" (where the result is too small to be stored in the specified type), up to "divide by zero" errors. In general, the former can be safely ignored, whereas the latter almost always causes a serious problem.

The way floating point calculations are executed is hardware-specific, but is often handled by a semi-independent "floating point machine". This reports errors in a different way to most other calculations, and makes it difficult to reliably catch and report errors at the point at which they occur. For example, some incorrect numerical operations don't raise an exception until the next calculation "flushes" the error buffer.

The controlling executable can request what floating point errors constitute errors, and roughly how they are handled. The default in Bladed up until 4.6 was to ignore all floating point errors, which meant that there are some external controllers which have been operating apparently correctly even when they contain divide-by-zero operations. The default behaviour was changed at 4.6 to exit when encountering floating point errors that could cause serious numerical errors if operation were to be allowed to continue. In Bladed 4.7+, this floating point protection can be switched off, to allow debugging.

Floating point exceptions are categorised in Bladed as being 'minor' or 'serious'. Minor exceptions are overlooked, whereas serious exceptions cause a run-time error.

- Divide by zero error - serious
- Floating point invalid - serious
- Floating point denormal number - serious
- Floating point overflow – minor, ignored
- Floating point underflow – minor, ignored
- Floating point inexact – minor, ignored

Floating point errors are one of the most common sources of external controller faults. The problem is confounded by them being very difficult to debug, and the fact that the controller may have worked (apparently) successfully in previous versions of Bladed.

Bladed 4.6+ makes an effort to trap and report serious floating point exceptions, and to date there have been no false-positive reports of floating point errors. The option to disable floating point protection in 4.7 should allow people to confirm whether this indeed the source of the failure.

Identifying the source of these exceptions is extremely difficult. Often the most effective way is to debug through the external controller, and instructions for this process can be found in the *Bladed External Controller User Manual* (version 4.6 or later), in Appendix A.

3.4 DLL Dependency Errors

Most modern programs use a combination of an executable together with supporting dlls (dynamic-link libraries). For the system to work, all of the dlls need to be of compatible versions exposing the correct functionality. The external controllers of Bladed, Tidal Bladed and WaveDyn are implemented as a dll which Bladed can talk to.

The Windows operating system manages the loading and use of dependent dlls, but report most errors back to the calling executable. Because of this, Bladed catches and reports most dll errors in the termination and message files. As Bladed has evolved, the error reporting has become more robust and comprehensive. Warnings are also reported in cases where there is a suspected problem, so it is always worth checking the message file contents.

3.4.1 Mixing the versions of dtbladed.exe and the ExternalControllerApi.dll

For non-legacy external controllers, there is a dependency on the ExternalControllerApi.dll, which is installed alongside Bladed. The executable should always be accompanied by its respective version of this dll. Incompatible versions can cause any number of problems, including program termination.

This problem can only occur if the Bladed installation is tampered with, or the executable is copied or moved without the correct version of the ExternControllerApi.dll.

Bladed 4.6+ will report a warning if the version of the dll does not match that of the executable.

3.4.2 Building the external controller with a non-compatible version of ExternalControllerApi.lib

The ExternalControllerApi.lib is required when compiling a non-legacy external controller. Bladed endeavours to be backwards compatible wherever possible, so controllers compiled with an older version of the lib should run fine. However, external controllers run with older versions of Bladed may suffer unknown problems.

It is not always possible to maintain backwards compatibility: for instance with functionality that is developing fast, and may require redefinition of the interface between versions. LIDAR would be an example of this.

3.4.3 A bad operating system call or dependency

Some more complex external controllers can themselves depend on other dlls – either created by the developer, or by a third party (e.g. standard Windows' libraries). If these are missing or incompatible, this will cause problems. It is the external controller developer's responsibility to ensure that the use of external functionality is trapped for errors, and all dependencies are distributed with the controller.

There are tools available to determine what dependencies a dll actually has, such as http://www.dependencywalker.com. These can be used to determine if a dependency has been created by accident during compilation.

These problems should be caught and reported clearly in Bladed 4.6+.

3.4.4 Incompatible C++ runtimes

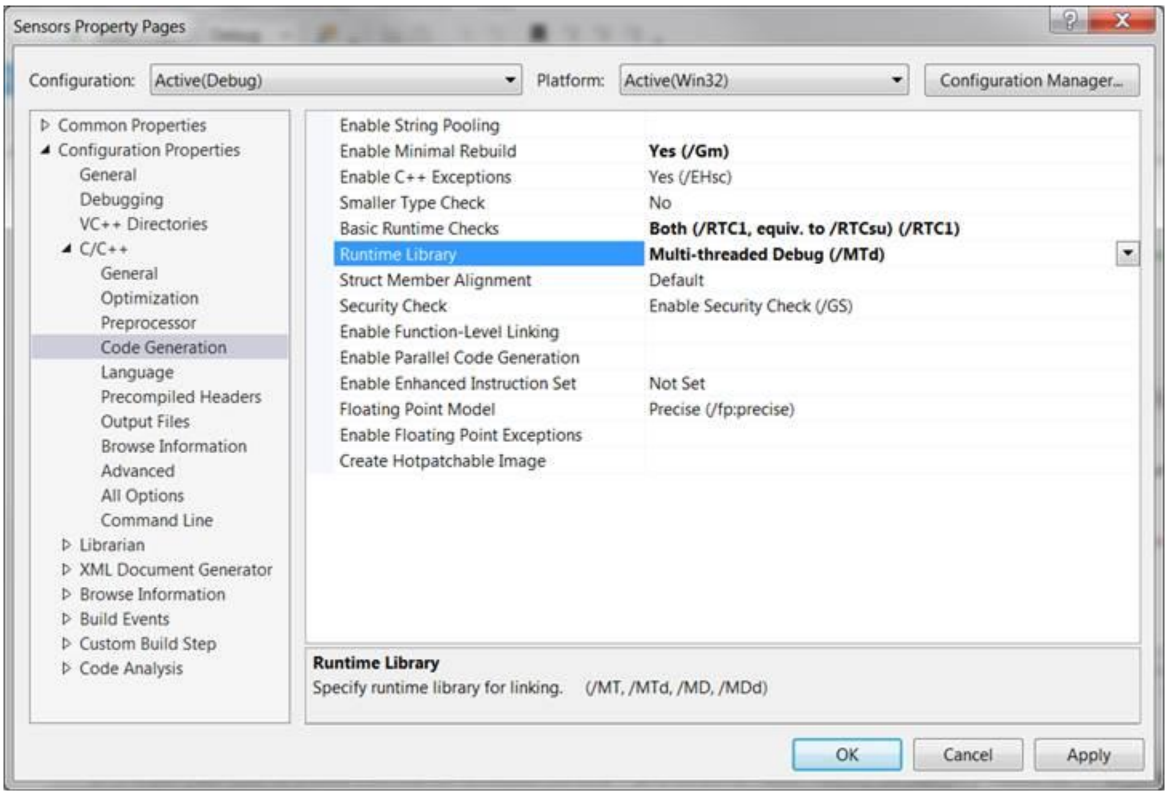
Bladed is compiled with a copy of the C++ runtime libraries. The version of this library is linked to which compiler is used, and this has changed during the development of Bladed 4.X. Due to this, Bladed 4.4+ uses a different version of the C++ runtime to versions 4.0 through to 4.3.

External controllers do not generally require the C++ runtime libraries, but some of the more complicated ones do. These libraries can either be compiled into the dll itself (as static libraries), or they can look on the machine at runtime for a compatible library. The default is generally the second as it is more efficient, but this allows the possibility of requiring a version that is different from that used by Bladed itself.

The recommended solution is to link the controller to a static version of the C++ runtime libraries, which removes the possibility of conflict. In Visual Studio, this can be done by going to the project's "Runtime Library" option at:

Properties->C/C++->Code Generation->Runtime Library

This specifies whether to link to an external copy of the C++ runtime, or include a static version in the dll itself:



Sensors Property Pages

Configuration: Active(Debug)   Platform: Active(Win32)

Enable String Pooling	
Enable Minimal Rebuild	**Yes (/Gm)**
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	**Both (/RTC1, equiv. to /RTCsu) (/RTC1)**
Runtime Library	**Multi-threaded Debug (/MTd)**
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Enable Function-Level Linking	
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	

Runtime Library
Specify runtime library for linking. (/MT, /MTd, /MD, /MDd)

Possible values are:

Multi-threaded Debug (/MTd)	Include the C++ runtime as a static library (Debug build).
Multi-threaded (/MT)	Include the C++ runtime as a static library (Release build).
Multi-threaded DLL (/MD)	Depend on MSVCR90.dll or MSVCR100.dll – which would need to match the Bladed version (risky)

Multi-threaded Debug DLL (/MDd)	Depend on MSVCR90d.dll or MSVCR100d.dll (this is not present on non-development machines).
Inherit from parent	This depends on what your default settings are.

This error is unusual, but far from unknown. This error should be caught and reported by Bladed 4.4+ as "*Windows Error 14001: the side-by-side configuration information for…*", which is the Windows description of an incompatible C++ runtime.

3.4.5 Using a controller compiled in Debug on a non-development PC

This is a variation of the above problem, where the debug build requires different versions of the standard libraries, usually MSVCR90D.dll instead of MSVCR90.dll, or MSVCR100D.dll instead of MSVCR100D.dll. A debug external controller will work fine on any machine where a C++ compiler is installed, but will fail on any other PC.

The above solution for 3.4.4 will fix this issue, but it is also recommended that only release versions of external controllers are distributed.