

Panic

Symbian OS uses **panics** to halt the flow of program execution. Unlike a leave, which can be trapped, if a panic occurs in an application running on the phone, the *application will be terminated*. This does not make for a good user experience, and, for this reason, panics should only be used in assertion statements, to halt code when an unrecoverable error occurs. Typically, these errors are *programmatic errors* and should be discovered and fixed before application shipment.

The Symbian SDK provides with a few assertion macros to verify code logic and program state. These are:

- **ASSERT**, which raises a "USER 0" panic *in debug builds* only
- **__ASSERT_DEBUG** and **__ASSERT_ALWAYS** allow you to specify what action to take if the condition fails, for debug only and for all builds respectively.

Or alternatively, we can always make use of **User::Panic()** within the code. This method panics the current thread, specifying a category name and panic number.

A few common panic codes are described below, the complete list of Symbian System Panics is located in the SDK help and online:

Symbian OS v9.x » Symbian OS reference » System panic reference[11]»

Contents

- [1 E32USER-CBase 63](#)
- [2 E32USER-CBase 69](#)
- [3 E32USER-CBase 71](#)
- [4 KERN-EXEC 3](#)
- [5 USER 13](#)
- [6 Miscellaneous panics](#)
- [7 Debug Panic Using Carbide C++](#)
- [8 Related Links](#)

E32USER-CBase 63

This panic is raised when an attempt is made to pop too many items from the cleanup stack for the current trap nest level.

```
HBufC* buffer = HBufC::NewL(15);
CleanupStack::PushL(buffer);
// Cleanup stack contains only 1 item, buffer.
// Pop 2 items from the cleanup stack -> E32USER-CBase 63 panic
CleanupStack::PopAndDestroy(2);
```

E32USER-CBase 69

All programs have their own cleanup stack, which they must create. E32USER-CBase 69 panic is raised if the cleanup stack is not created before using it. Here's an example:

```
GLDEF_C TInt E32Main()
{
    // This raises E32USER-CBase 69 panic
    HBufC* buffer = HBufC::NewLC(10);

    return KErrNone;
}
```

E32USER-CBase 71

Anything that is pushed onto the cleanup stack inside a trap harness must be popped off before leaving the harness. E32USER-CBase 71 panic is raised when there are several nested trap harness levels, and an attempt is made to exit from one level before all the cleanup items belonging to that level have been popped off the cleanup stack. A common action that causes this panic is trapping an ?LC? method. Here's an example:

```
TRAPD(error, pointer = CExample::SomeFunctionLC());
```

E32USER-CBase 71 panic is raised upon returning from the trap harness because SomeFunctionLC has placed pointer onto the cleanup stack, where it remains upon exiting this particular harness level. The panic can be avoided by popping the object from within the trap harness:

```
TRAPD( error, pointer = CExample::SomeFunctionLC();
    CleanupStack::Pop(pointer) );
```

Now, if SomeFunctionLC leaves, pointer is destroyed as part of leave processing. If the function does not leave, then CleanupStack::Pop(pointer) is called, which avoids the panic.

Note: It is not recommended to call an LC function from inside a trap harness.

KERN-EXEC 3

KERN-EXEC 3 means a general unhandled exception. In other words, it is quite difficult to know what causes it. In this section, some suggestions are presented. A reference to an uninitialized variable:

```
CConsoleBase* console;
// Console initialization is commented out, causing a KERN-EXEC 3 panic
// console = Console::NewL(KTextConsoleTitle,
//    TSize(KConsFullScreen, KConsFullScreen));
console->Printf(KTextHello);
```

Too large a value in a formatting string that contains a pointer to a descriptor (?%S?):

```
_LIT(KUnitTxt, "units");
HBufC* unitTxt = KUnitTxt().AllocLC();
// The integer is too big to fit in "%d"; "%Ld" should be used instead.
TInt64 totalSize = 12345678901;
```

Panic

```
console->Printf(_L("Total size is %d %S"), totalSize, unitTxt);  
CleanupStack::PopAndDestroy(unitTxt);
```

USER 13

USER 13 panic is raised if an invalid variable list is passed to the AppendFormatList function when the format is %S or %s. In the following example, the panic is raised by _L macro, because %d contains only the first 32 bits of the 64 bit totalWeight. This causes the remaining 32 bits flow over and be considered as a pointer because of the %S formatting character.

```
_LIT(KUnitTxt, "kg");  
// TInt64 type requires %Ld or %Lu as the formatting character, not %d  
TInt64 totalWeight = 150;  
console->Printf(_L("Total weight is %d %S"), totalWeight, &KUnitTxt);  
Note: This panic is raised only in debug builds.
```

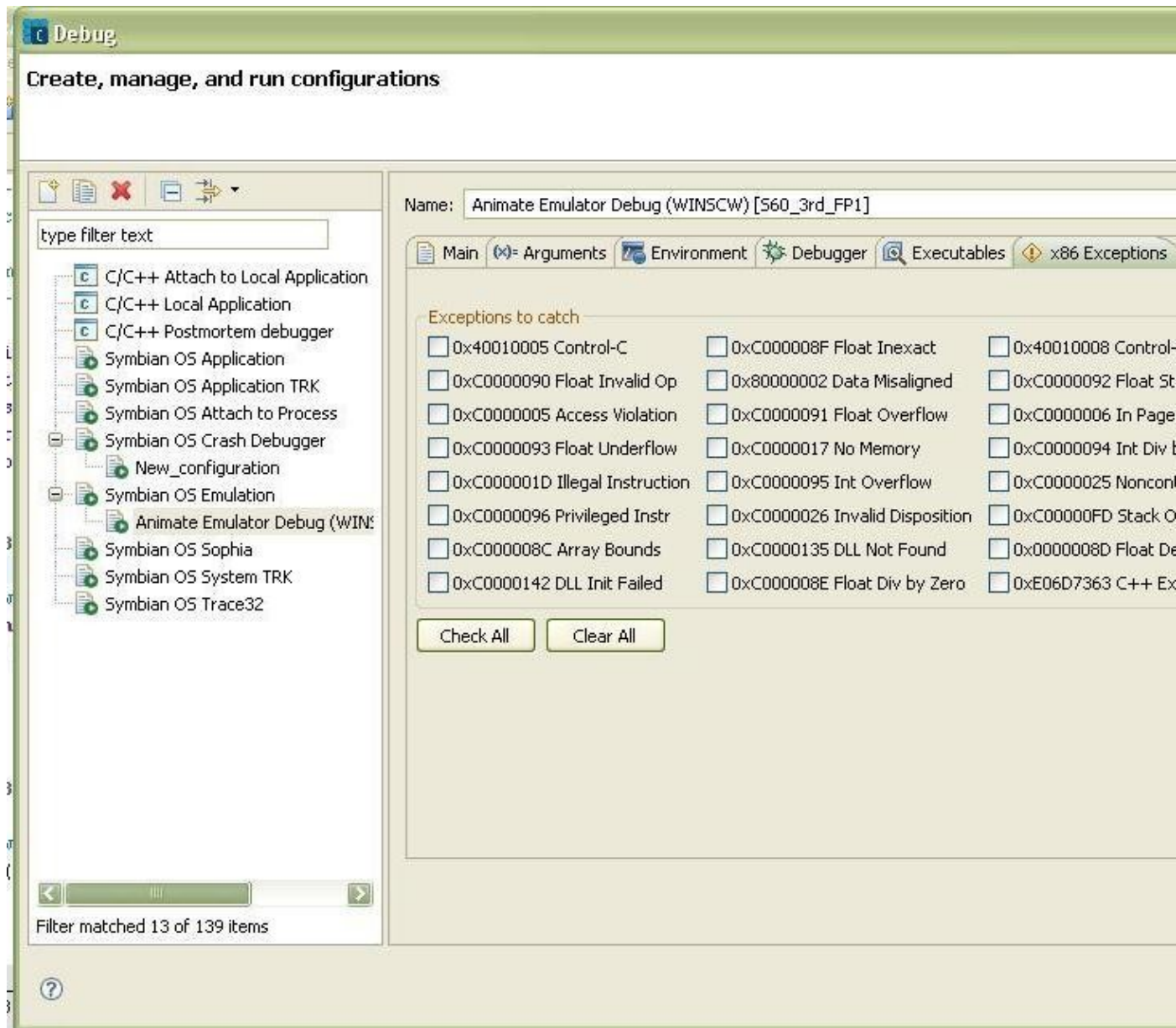
Miscellaneous panics

Panic code	Reason	Solution
E32USER-CBase 90	An item to be popped is not the expected item.	Check the call to CleanupStack::Pop
USER 0	An assertion has not been met.	Check the assertion.

Debug Panic Using Carbide C++

Open the Carbide C++ Debug Dialog. Check on the Option below. This will help to debug various panics like KERN - EXEC - 3, USER 130 etc very easily. Last option does work in 3rd edition so just uncheck it . :)

Panic



Related Links

- [Symbian OS System Panic Reference](#)