



## Contents

- [1 Why use GCCE 4](#)
- [2 Disclaimer](#)
- [3 Necessary files](#)
- [4 Versions](#)
- [5 Replacement of the toolchain](#)
  - ◆ [5.1 Backup the old toolchain](#)
  - ◆ [5.2 Copy the new toolchain](#)
- [6 Patching the SDK](#)
  - ◆ [6.1 Edit the compilation scripts](#)
  - ◆ [6.2 Remove failing assertions](#)
  - ◆ [6.3 Correct va\\_lists](#)
  - ◆ [6.4 Remove extra qualifiers from the SDK's header files](#)
  - ◆ [6.5 Solve linker errors for libsupc++. Detected with GCCE 4.3.2 only](#)
  - ◆ [6.6 Supply your own integer division routines](#)
- [7 Project cleaning](#)
- [8 Known migration issues](#)
  - ◆ [8.1 Warnings](#)
  - ◆ [8.2 Static initialization fiascos](#)

## Why use GCCE 4

The SDKs for Symbian OS v.9.1+ development rely on the GCCE CSL Arm Toolchain to compile code for the real devices. However, the version of GCCE compiler distributed with the SDKs is quite old, 3.4.3. - originally published in November 2004.

Although GCC 3.4.3 is, in most cases, a good compiler, it nevertheless contains some problems. It is also quite slow. During the development of bigger applications, you may find out that a newer toolchain is preferable for the following reasons:

- GCC 4 compiler can be noticeably faster, especially in C code.
- GCC 4 seems to have fewer problems with optimisation.
- GCC 4 produces significantly smaller binaries (for example, in the testcase, the SIS with the full application shrank from 1.4 MB to 1 MB (Release)).
- GCC 4 seems to have fewer miscompilations.
- GCC 4 is stricter in producing errors and warnings, therefore resulting in better code quality.

## Disclaimer

If you are planning to use GCCE 4 in your SDK, note that no functionality or correctness can be guaranteed. This is a "best-effort" HOWTO, provided with absolutely no warranty.

According to [CodeSourcery](#):

"You are attempting to do something that we do not recommend you do with the zero-cost downloads. The Lite toolchains are provided at zero cost without support."

If you want to be safe, make a backup before you start.

## Necessary files

First, a new version of the CSL Arm Toolchain must be downloaded from [CodeSourcery](#). The path for downloading is [here](#). If the direct link does not work, you can find the download from the 'Downloads' menu.

CodeSourcery produces many different packages. The package you need is "Sourcery G++ Lite Edition for ARM", with Symbian OS as the target OS. The package should be available free of charge.

There is also an archive of previous releases available. The releases are usually made twice a year. It is often recommended to choose an older release.

## Versions

All packages have been tested with SDK for Series 60 3rd Edition, Maintenance Release (Symbian 9.1)

Package version	GCC version	Results of tests
2006q1 - 6	4.1.0	Even in simple code, a compilation error with "internal compiler error" message. Recommendation: do not attempt to use it.
2007q1 - 26	4.2.0	After making necessary changes to the SDK, the compilation runs well. HelloWorldBasic can be built and run on device (both Debug and Release modes).
2007q3 - 52	4.2.1	Not tested. Should behave similarly to 4.2.0 and 4.2.3.
2008q1 - 126	4.2.3	After making necessary changes to the SDK, compilation runs well.  HelloWorldBasic can be built and run on device (both Debug and Release modes). This version has been tested on a very large project (over 100,000 lines of code): some changes were necessary, but overall it seemed to work.
2008q3 ? 67	4.3.2	After making necessary changes to the SDK, the compilation runs well.  Linker problems with missing "_Unwind_GetTextRelBase". Recommendation: stick to 4.2.3, if possible.
2009q1 - 162	4.3.3	After making all the changes listed in this article, the

## Replacement of the toolchain

After downloading the desired package version from CodeSourcery, install it. By default, it will install itself into the following directory:

```
c:\program files\CodeSourcery\Sourcery G++ Lite\
```

## Backup the old toolchain

Before you start altering the toolchain, make a backup of the old version (with 3.4.3). In the case of a problem, you will not then need to reinstall the SDK.

The version supplied with the SDKs usually resides in the following directory:

```
C:\Program Files\CSL Arm Toolchain\
```

Create another directory, for example C:\Program Files\CSL-Backup\ and copy the whole content of the old version directory to the backup directory. If anything goes wrong, you will simply need to delete the altered contents of the C:\Program Files\CSL Arm Toolchain\ directory, and copy the backup back to its original location.

## Copy the new toolchain

Remove the entire old content of the C:\Program Files\CSL Arm Toolchain\ directory and copy the content of the new toolchain directory from c:\program files\CodeSourcery\Sourcery G++ Lite\ into the old directory.

Having done this, you have the new toolchain in place. However, you still need to patch your SDK in order to work with the new toolchain.

## Patching the SDK

### Edit the compilation scripts

As far as known, all SDKs require this patch.

There are some files in the epoc32\tools subdirectory of the SDK which refer to the exact version of the toolchain, and therefore need patching. Namely:

```
cl_bpabi.pm  
cl_gcce.pm  
ide_cw.pm  
compilation_config/gcce.mk
```

## How\_to\_use\_GCCE\_4\_with\_Symbian\_SDKs

Backup the above files into a different directory (for example, "original\_343"). Then edit all the files, and removing the "3.4.3" strings and substituting them with the GCC version you are using in the new toolchain (for example, "4.2.3").

Backup the files into a **different directory** - do not make backups in the same directory. The script which processes them in the IDE can mistake the backups for the real files. See [the discussion here](#) for more information.

## Remove failing assertions

This patch concerns SDK for Series 60 3rd Edition, Maintenance Release. Your SDK may vary, and possibly may not need this patch at all.

Find the file d32locd.h in your SDK's include directory. Make a backup of this file. Then find and comment out the following lines:

```
__ASSERT_COMPILE(_FOFF(TLocalDriveCaps,iSize)%8 == 0);  
__ASSERT_COMPILE(_FOFF(TLocalDriveCapsV3,iFormatInfo.iCapacity) % 8 == 0);
```

These lines would trigger a compilation error with GCC 4 ("not a constant", etc.)

## Correct va\_lists

This patch concerns SDK for Series 60 3rd Edition, Maintenance Release. Your SDK may vary, and possibly may not need this patch at all.

The type

```
va_list
```

and related types are used in processing functions that have variable argument length (for example,

```
printf
```

function from the LIBC library). After the installation of the new toolchain, you will have multiple definitions of the necessary types, which will confuse the compiler into printing out errors each time you use

```
va_list
```

and related types. To avoid this, find the file

```
gcce\gcce.h
```

in your SDK's include directory. Make a backup of this file. In this file, find and comment out the lines:

```
typedef struct __va_list { void *__ap; } va_list;  
#define va_start(ap, parmN) __builtin_va_start(ap.__ap, parmN)  
#define va_arg(ap, type) __builtin_va_arg(ap.__ap, type)  
#define va_end(ap) __builtin_va_end(ap.__ap)  
typedef va_list __e32_va_list;
```

Edit the compilation scripts

## How\_to\_use\_GCCE\_4\_with\_Symbian\_SDKs

and add the following line just before the first commented-out line:

```
#include <libc/stdarg.h>
```

Nota that because of this patch, each of your projects now de facto include the stdarg.h header from the standard C library. To contradict this, you must have the following line in the MMP file(s) of the project(s) you will be compiling with the new toolchain:

```
SYSTEMINCLUDE      \epoc32\include\libc
```

since the

```
stdarg.h
```

file includes more files from that directory.

## Remove extra qualifiers from the SDK's header files

This patch concerns SDK for Series 60 3rd Edition, Maintenance Release. Your SDK may vary, and possibly may not need this patch at all.

If you are using AVKON Query Dialog, find aknquerydialog.h in your SDK's include directory. Make a backup of this file. Then, find the following line:

```
CCoeControl* CAknQueryDialog::FindControlOnAnyPageWithControlType(TInt aControlType, TInt* aLineI
```

and remove the qualifier

```
CAknQueryDialog::
```

so that the line looks as follows:

```
CCoeControl* FindControlOnAnyPageWithControlType(TInt aControlType, TInt* aLineIndex=0, TInt* aPa
```

This prevents a compiler error from "extra qualification", which did not result in error in GCC 3.4.3, but is detected by GCC version 4.

If you are using ImageConversion.h, you will meet a similar problem in it. The offending line is

```
IMPORT_C static CIclRecognizerUtil* CIclRecognizerUtil::NewL();
```

Again, remove the extra qualification.

You may find more "extra qualification" errors in Symbian SDK include files. It seems to be a common problem. Next such error is in mmf\mmfcontrollerpluginresolver.h, then coectrl.h, etc. They must all be patched. It should not be more than 4-5 errors.

The same applies to your own code as well - no extra qualifiers are allowed.

## Solve linker errors for libsupc++. Detected with GCCE 4.3.2 only

During the linking phase, the GCCE 4.3.2 toolchain will complain about missing reference to `<pr>_Unwind_GetTextRelBase</PR>`.

This is caused by missing binary code in

```
CSL Arm Toolchain\lib\libsupc++.a
```

This file has about 7 kB in the 4.3.2 distribution.

There are two working solutions to this problem.

A simple solution is to substitute this file with the original file from some older toolchain (even 3.4.3). That file has about 15 kB. Just rewrite the newer file with the older file.

A more complicated solution is to write the missing functions yourself and add them to the source code of your project. The functions can be empty. This solution has not been tested here.

This problem was not detected with GCCE 4.2.0 or 4.2.3. So the solution might be to use them instead of 4.3.2.

## Supply your own integer division routines

Some of the SDKs for Series 60 3rd Edition suffer from the following problem: the system libraries do not contain definitions of compiler helper functions `__aeabi_uidiv` and `__aeabi_idiv`. This was an omission on the part of Symbian, and it is not limited to the SDKs - the devices are affected as well.

If you do not intend to ever use integral division in your project, you may skip this section entirely.

To find out whether your SDK contains this error, enter the following code snippet somewhere in your code (not in an unreachable part), and try compiling the code with GCCE DEBUG:

```
#include <e32debug.h>

...
void tryDivision ()
{
    TInt a, b, c;
    a = 10;
    b = 5;
    c = a / b;
    RDebug::Printf("Result of division is %d",c);
}
```

Next, call the `tryDivision()` function somewhere in your code, for example in `ConstructL` method of the `AppUi` class.

(The `Printf` line is necessary here because otherwise the compiler may throw out the useless division code from your binary, and you will be unable to detect possible errors.)

## How\_to\_use\_GCCE\_4\_with\_Symbian\_SDKs

The code will always be compilable in WINSCW. However, in GCCE DEBUG, the build process may abort with the message "missing reference to \_\_aeabi\_idiv"

In this case, you have an incorrect SDK, and you will have to perform the following fix.

Create a new file in your project's src/ directory, named division.c. Don't use the extension .cpp because this will not work! The content of the file will be as follows:

```
// This code was suggested by Julian Brown from CodeSourcery. It is in public domain.
// Many thanks!

#if __GCCE__
#if __SERIES60_30__
extern unsigned int __aeabi_uidivmod(unsigned numerator, unsigned denominator);
int __aeabi_idiv(int numerator, int denominator)
{
    int neg_result = (numerator ^ denominator) & 0x80000000;
    int result = __aeabi_uidivmod ((numerator < 0) ? -numerator : numerator, (denominator < 0) ?
    return neg_result ? -result : result;
}
unsigned __aeabi_uidiv(unsigned numerator, unsigned denominator)
{
    return __aeabi_uidivmod (numerator, denominator);
}
#endif // __SERIES60_30__
#endif // __GCCE__
```

The file will be automatically added to your MMP file. Adjust the preprocessor blocks (#if \_\_SERIES60\_30\_\_) as needed.

## Project cleaning

Run the following commands from the command in your project's group directory:

```
bldmake clean
bldmake bldfiles
abld build gcce udeb
```

This should clear and recreate your GCCE makefiles (which contain paths to the old toolchain).

## Known migration issues

### Warnings

GCCE 4 issues more warnings than GCCE 3 and as a result, you will find many warnings in previously warning-free code. This is, however, a feature rather than a bug as most of the warnings are useful.

## Static initialization fiascos

GCCE 4 is more sensitive to "static initialization fiasco" than GCCE 3.

"Static initialization fiasco" means a situation where you initialize one static variable (or constant) using another static variable (or constant). Since the compiler or system does not guarantee the correct sequence of static initializations, you will encounter errors if you try to initialize something with a yet-uninitialized value.

To read more about the "static initialization fiasco", visit the [C++ Faq Lite](#).

If you have "static initialization fiascos" in your Symbian code, chances are that GCCE 3.4.3 will not trigger any errors, while GCCE 4.x will. That is because the compilers produce different code, and the static initialization sequence will therefore differ.

This is an extremely frustrating situation, as it results in premature exits of your application. However, static initialization fiasco is a serious problem and which will not go away if you return to older GCCE. Chances are that once, in the future, if you need to compile your code with newer toolchain, you will be forced to do the repairs anyway.

A typical example is a static const descriptor initialized from another static const descriptor, or a static const struct, which uses static const descriptors, or a static const struct that uses resource ids. Solution: in this case, the offending structure is often used only locally, in a single class or a single function. It is therefore worth considering moving it there from a global header file.

The way how to detect "static initialization fiascos" is to run on/device debugging with a GCCE DEBUG build and watch for errors. The panic stack will navigate you to the precise structure which caused the problem (click on the "static initialization and destruction" line of the stack).