Code Coverage Using gcov

Adrian Kosmaczewski

2007-07-23

I've just uploaded a code coverage test project, using the gcov GNU tool. The idea was to create a small application (simulating an ATM), and injecting into the CppUnit unit tests executable code coverage information, using the gcov utility. And the results just speak by themselves:

```
      32:  301:    const bool Date::isLeapYear() const
       -:  302:    {
      32:  303:        if ( _year % 4 != 0 )
       -:  304:        {
      19:  305:            return false;
       -:  306:        }
       -:  307:        else
       -:  308:        {
      13:  309:            if ( _year % 100 != 0 )
       -:  310:            {
   #####:  311:                return true;
       -:  312:            }
       -:  313:            else
       -:  314:            {
      13:  315:                if ( _year % 400 != 0 )
       -:  316:                {
       2:  317:                    return false;
       -:  318:                }
       -:  319:                else
       -:  320:                {
      11:  321:                    return true;
       -:  322:                }
       -:  323:            }
       -:  324:        }
       -:  325:    }
```

The above code sample from the date.cpp.gcov file (in the project zip file) show that the unit tests run called 32 times the Date::isLeapYear() method, of which 19 were not leap years, and the rest were. The interesting bit is in line 311, which was never called, as shown with the "#####" sign! This is extremely

nice, since it shows that my tests are not 100% comprehensive, and some cases are not tested.

On the downside, I must say that the gcov tool is not really easy to use (it took me a while to figure out how to do things) but grosso modo it works in the following way:

1. You must compile and link the unit test application (for example in "Debug" mode, or maybe other special ad hoc configuration) using the `-fprofile-arcs -ftest-coverage` GCC flags (in Xcode you can check the "Generate Test Coverage Files" and "Instrument Program Flow" option checkboxes) and the `-lgcov` linker flag. I also set Xcode to "ZeroLink", told GCC to do optimization level "None [-O0]", unchecked the option to generate position-dependent code, and disabled prebinding;
2. You run the application: this will generate a folder with a bunch of files with ".gcda" and ".gcno" extensions, together where the `.o` files are output during the build (when using Xcode on a PPC Mac, these files are created in build/atm.build/Debug/tests.build/Objects-normal/ppc);
3. Open a command line window and run the commands `gcov FILENAME.gcno` and `gcov FILENAME.gcda` and you will get, in the same folder, a file called `FILENAME.cpp.gcov` with the information shown in the snippet above. By the way, you get files for all the dependencies of that source code as well.

I hope this helps! Having this information can really help ensuring that test cases are comprehensive and complete. As always, if someone has a tip or a comment about gcov and wants to share it, I would be glad to read about it in the comments below.