# GnuCOBOL 2.0 Release Candidate 2 [06NOV2016] Build Guide for MinGW - Minimalist GNU for Windows

cobc (GnuCOBOL) 2.0.0

Written by Keisuke Nishida, Roger While, Ron Norman, Simon Sobisch, Edward Hart.

This document was prepared by: Arnold J. Trembley (arnold.trembley@att.net)
and last updated Friday, 02 June 2017.

Please refer to the excellent manual written by Gary Cutler (CutlerGL@gmail.com):

**OpenCOBOL-1.1-06FEB2009-Build-Guide-For-MinGW.pdf**

http://www.arnoldtrembley.com/OpenCOBOL-1.1-06FEB2009-Build-Guide-For-MinGW.pdf

I used Gary Cutler's document as a guide when building GnuCOBOL 1.1 and GnuCOBOL 2.0 using MinGW. This document will be much less detailed, and will describe how to build GnuCOBOL 2.0 rather than older versions of OpenCOBOL or GnuCOBOL.

**Simon Sobisch** of the GnuCOBOL project on Sourceforge.net was extremely helpful whenever I encountered difficulties building GnuCOBOL, especially in running the compiler test suites and VBISAM. **Brian Tiffin** also provided assistance and encouragement. The GnuCOBOL/OpenCOBOL project can be found at:

https://sourceforge.net/projects/open-cobol/

**GnuCOBOL 2.0 Build Guide for MinGW (draft)**

**Required Components:**

Based on Gary Cutler's build guide, you will need to find and install the following components in order to build the GnuCOBOL 2.0 compiler in Windows:

1. MinGW - Minimalist GNU for Windows
2. GNU Multiple-Precision Arithmetic package (gmplib)
3. PDCurses 3.4 - used for screen read and write.
4. Berkeley Database (BDB) package from Oracle ==OR==
5. VBISAM2 by Roger While, from Sourceforge
6. GnuCOBOL 2.0 source code from the OpenCOBOL project on Sourceforge.net

You may want to download all these packages first and make your own backups before starting the GnuCOBOL build process.

**Licensing:**

This document will not discuss open source licensing in detail. Please refer to Gary Cutler's document. The GnuCOBOL 2.0 compiler is licensed under the GNU General Public License (GPL) version 3, and the runtime libraries are licensed under the GNU Lesser General Public License (LGPL) version 3. The Oracle Berkeley Database (BDB) package, used for indexed sequential file access, has some license restrictions related to distribution of compiled GnuCOBOL programs that could require distributing your COBOL source code or else paying a license fee to Oracle. There are no similar license restrictions if the VBISAM package is used for indexed sequential file access (instead of BDB), or if no indexed sequential file access will be supported (NODB).

**Download the packages:**

The **MinGW** software package can be downloaded from:
https://sourceforge.net/projects/mingw/files/

Download the file named "mingw-get-setup.exe". This should be the 32-bit version of MinGW. MinGW is a Unix-like environment for Windows needed to run GCC (the GNU Compiler Collection) to build the GnuCOBOL compiler. The GnuCOBOL compiler will translate COBOL source code into C source code, and the embedded MinGW GCC compiler will translate the intermediate C code into executable code.

**GnuCOBOL 2.0 Build Guide for MinGW (draft)**

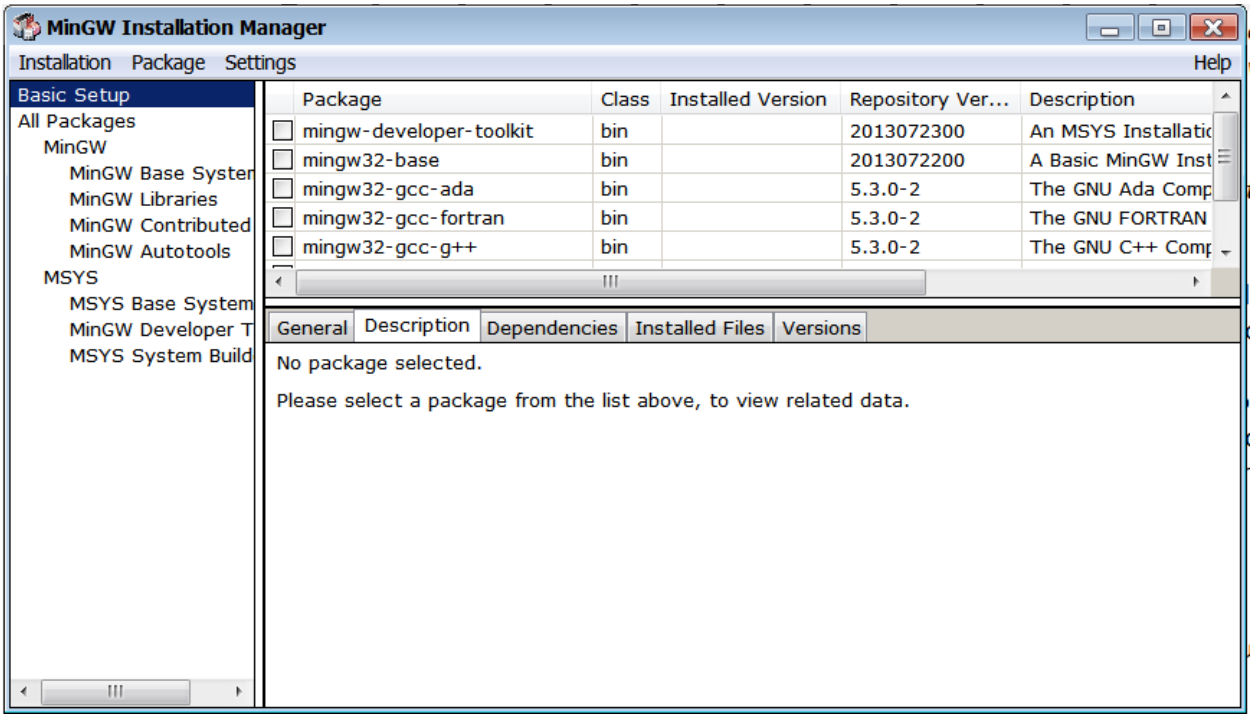
## Installing MinGW and the MSYS bash shell

First, create a wrapper folder to hold MinGW and keep it separate from any other MinGW installation on the same PC. For this I created **C:\GC20-build\MinGW.** When you run "mingw-get-setup.exe" it will default to installing in **C:\MinGW** and you should override this with **C:\GC20-build\MinGW**. The folder must already exist in order to be selected.

Continue with "mingw-get-setup.exe" and let it download the catalog of current components. When that is complete, use the GUI interface to select the "mingw32-base" and "msys-base" packages from the "Basic Setup" menu.

Then select the following components from the "All Packages" menu:

```
mingw32-gettext.bin      /* for translatable strings */
mingw32-gettext.dev      /* for translatable strings */
mingw32-gettext.dll      /* for translatable strings */
MinGW Autotools          /* for NIST COBOL85 testing */
msys-help2man.bin        /* for GnuCOBOL make */
msys-m4.bin              /* for builing Berkeley DB */
msys-perl.bin            /* for NIST COBOL85 testing */
msys-wget.bin            /* for NIST COBOL85 testing */
```



Then have “mingw-get” apply all changes that were previously marked. The installation may take 5 to 15 minutes.

Immediately after installing **C:\GC20-build\MinGW,** copy the **cc1.exe** file to an additional location:

Copy "C:\GC20-build\MinGW\libexec\gcc\mingw32\5.3.0**cc1.exe**"
to "C:\GC20-build\MinGW\bin**cc1.exe**".

In the example above, “5.3.0” is the version of GCC in MinGW. This version number may be different when you download the “mingw-get” GUI interface installer.

Next, verify that **C:\GC20-build\MinGW\bin** contains a file named **mingwm10.dll.**

Then verify that a directory named "**C:\GC20-build\MinGW\MSYS\1.0\etc**" exists and that it contains a file named "**fstab**" in it with no file extension. View this file with Notepad or Wordpad and verify that it has the following line in it:

```
c:/GC20-build/MinGW /mingw
```

This “fstab” file may contain comment lines which begin with “#”, and they can be ignored. Since Unix file and folder names are case-sensitive, this fstab file tells MinGW/MSYS to treat the Windows "**C:\GC20-build\MinGW**" folder as the Unix/Linux mount point named “/mingw”.

```
# /etc/fstab -- mount table configuration for MSYS.
# Please refer to /etc/fstab.sample for explanatory annotation.

# MSYS-Portable needs this "magic" comment:
# MSYSROOT=C:/MinGW/msys/1.0

# Win32_Path                    Mount_Point
#-----------------------------------------------
C:/GC20-build/MinGW             /mingw
```

MSYS will only use the “/mingw” mount point, even though it is "**C:\GC20-build\MinGW**" to Windows. If you selected a different folder on the MinGW startup screen, it should be automatically built into the “fstab” file.

Finally, you will need to create a shortcut on your windows desktop to:

```
C:\GC20-build\MinGW\MSYS\1.0\msys.bat -norxvt
```

Do not forget the “-norxvt” parameters in the shortcut. This tells the msys.bat file to use the “sh” shell instead of the “rxvt” shell. You may also want to change the icon for the shortcut to use “C:\GC20-build\MinGW\MSYS\1.0\msys.ico”.

You should now have an icon "**MSYS**" on your desktop. Double-click it to start it.

**Configure the BASH shell, from Gary Cutler's document:**

*The window started by the "MSYS" icon resembles a Windows console, but is actually an MSYS "bash" shell; Use the window's "Properties" command as you would do with a normal window to change the window size to 190 columns by 60 rows - make sure the buffer size has a "height" of at least 600; you'll also want to change the font to something that enables that window to fit on your screen (I use "Lucida Console" with a font "size" of "10").*

*If you are running Windows Vista or Windows 7, close the bash window and restart it again, this time giving it Administrator authority via "**Run As Administrator**".*

I strongly recommend changing the font type and size first (lucida console 10), and then changing the MSYS window to 190 columns by 60 rows, and making the "height" at least 2000 lines (or even more). When running "make" commands many components display thousands of messages in the MSYS window, and you may want to scroll back to view them.

If you are running MinGW on a widescreen laptop you may want to limit the window height to 50 or 55 rows for your convenience. You should also set the MSYS properties “edit options” to enable both “quick edit mode” and “insert mode”. That will allow you to paste commands into the MSYS bash shell using the right mouse button (instead of Ctrl-V), and copy blocks of text messages from the MSYS bash shell window and paste them into other documents.

### Unexpected Errors

If unexpected errors occur anytime during the GnuCOBOL build process, first check that all steps are run in the correct order and no steps are skipped. For example, accidentally omitting any "make install" command will prevent the final GnuCOBOL from building. Then check that all MSYS commands have been entered correctly, without any typing mistakes.

It is still possible to have unexpected errors and for those you should join the GnuCOBOL forum "Help Getting Started" and ask questions there. Be prepared to show exactly what errors occurred. It will also help if you register with the OpenCOBOL Sourceforge project, so your posts can appear immediately without having to wait to be moderated.

Here are the links to the Sourceforge project and the discussion forums:

https://sourceforge.net/projects/open-cobol/files/nist/

https://sourceforge.net/p/open-cobol/discussion/

### Anti-Virus Considerations

If your anti-virus system tries to block any programs generated by MinGW GCC, you may want to disconnect from the internet and disable your anti-virus protection. Or you may want to exclude the entire C:\GC20-build\MinGW* folder from anti-virus scanning if that is an option. I also had to create exclusions for dummy.exe in the Windows %User%\temp folder.

I found that McAfee Antivirus and Windows Defender/Microsoft Security Essentials did not appear to have conflicts with MinGW GnuCOBOL builds, while freeware Avast Anti-virus required exclusion rules. One user in the GnuCOBOL/OpenCOBOL Sourceforge forum reported that Norton Anti-virus quarantined MinGW output as possible viruses.

If you intend to run the NIST COBOL85 test suite, you either need an internet connection to download the "**newcob.val**" file during testing, or you need to download it from Sourceforge before building the components:

https://sourceforge.net/projects/open-cobol/files/nist/

You will need to choose one of the archive files (newcob.zip, newcob.7z, or newcob.val.tar.gz), and expand it to extract the "**newcob.val**" file. It is 26 megabytes and contains all the NIST COBOL85 test programs.

## Building GMP (GNU Multiple-Precision Arithmetic Library)

The **GMP 6.1.2** GNU Multiple-Precision Arithmetic Library can be downloaded from:
https://gmplib.org/#DOWNLOAD

As of March, 2017, The most current GMP version is in a file named "gmp-6.1.2.tar.lz", but previous versions of MSYS could not unpack a "tar.lz" file. I used the free "lzip.exe" for windows to convert "gmp-6.1.2.tar.lz" into "gmp-6.1.2.tar".

Lzip.exe can be downloaded as "lzip-1.11" from:
http://www.softpedia.com/get/Compression-tools/Lzip.shtml

Once you have decompressed "gmp-6.1.2.tar.lz" into "gmp-6.1.2.tar" you will need to copy the "tar" file into your /mingw/MSYS folder using Windows Explorer or the Windows CMD.EXE command shell. Then use the MSYS bash shell to execute the following commands to unpack the gmp tarball:

```
cd /mingw/MSYS
tar xf gmp*.tar      /* unpack non-compressed tar */
rm gmp-*.tar         /* remove the "tar" file  */
```

At this point you have a Windows folder named C:\GC20-build\MinGW\MSYS\gmp-6.1.2. The next five MSYS commands will build compile gmp 6.1.2 for us:

```
cd gmp*
./configure --prefix=/mingw --enable-shared --disable-static
make
make check          /* Test gmp 6.1.2*/
make install
```

You may find it easier to copy and paste these commands into the MSYS bash shell one line at at time, especially for the complicated commands. The four commands to build GMP take 10 to 20 minutes to run. Commands can also be concatenated using the && operator, for example “make && make check && make install”. Processing will stop early if any errors are found.

The "./configure" command took a long time to run and produced hundreds of messages.

The "make" command took a very long time to run and produced thousands of messages.

The "make check" command ran for a very long time and produced thousands of messages. It performs hundreds of tests against the generated gmplib components

The "make install" command runs very quickly. It can be run before “make check”, but do not forget to run “make install” to prevent problems building the final COBOL compiler.

After "make install" completes, run the following command in MSYS to verify that a file named "**/mingw/bin/libgmp-10.dll**" exists:

```
ls /mingw/bin/libgmp*.dll
```

## Building PDCurses

The **PDCurses 3.4** package is in a file named "**PDCurses-3.4.tar.gz**" which can be downloaded from:
https://sourceforge.net/projects/pdcurses/files/pdcurses/3.4/

The PDCurses package is used for COBOL SCREEN-SECTION and extended console input-output support (DISPLAY/ACCEPT AT/WITH). This package did not need any changes from Gary Cutler's original instructions. As of March, 2017, PDCurses 3.4 built on 2008-09-08 is still the most current version.

Copy "PDCurses-3.4.tar.gz" into your **C:\GC20-build\MinGW\MSYS** folder using Windows commands, and then use the following bash shell commands to uncompress it:

```
cd /mingw/MSYS
tar xzf PDC*.tar.gz
rm PDC*.tar.gz
```

Then build PDCurses using these commands:

```
cd PDC*/win32
make -f gccwin32.mak DLL=Y
```

Then use the following command to verify that a "pdcurses.dll" exists in the "**C:\GC20-build\MinGW\MSYS\PDCurses-3.4\win32**" folder.

```
ls *.dll
```

You can also verify this using Windows Explorer.

Gary Cutler's document said "*there is no 'make install' available for PDCurses, so you'll have to deploy the PDCurses components yourself*". Here are the MSYS commands to do that, and this is another instance where it may help you to paste these commands into the MSYS bash shell window:

```
cp pdcurses.dll /mingw/bin/.
cp pdcurses.a /mingw/lib/libpdcurses.a
cd ..
cp *.h /mingw/include/.
cp curses.h /mingw/include/pdcurses.h
```

## Process Improvement Checkpoint

At this point in the build process I normally backup the entire "c:\GC20-Build" folder so that I can restore it with gmplib and PDCurses already built. This saves me a lot of time when building multiple versions of GnuCOBOL with either Berkeley DataBase, no indexed sequential file support, or VBISAM 2.0. The backup folder includes the MinGW get program and the shortcut to MSYS:

```
 Directory of C:\GC20-Build-BKUP

11/09/2016  07:47 PM    <DIR>          .
11/09/2016  07:47 PM    <DIR>          ..
11/09/2016  07:48 PM    <DIR>          MinGW
09/07/2014  02:53 AM            86,528 mingw-get-setup.exe
11/09/2016  06:44 PM             2,211 msys.bat - Shortcut.lnk
               2 File(s)         88,739 bytes
```

For example, if I have already built GnuCOBOL with BDB and I want to build another version with VBISAM 2.0, I simply restore the C:\GC20-Build folder from the backup and copy the MSYS shortcut to the desktop. Then I can start MSYS with gmplib and PDCurses already built.

Backing up the C:\GC20-Build folder is completely optional at this point. But if you build GnuCOBOL frequently then restoring the build folder can save you an hour or two on every subsequent build.

It should be possible to use MinGW get to install pre-built gmplib and curses support, but my first and only attempt at this was not successful. One of my goals is to test building GnuCOBOL with pre-built MinGW gmplib and curses support. If that can be made to work reliably, then these instructions can be simplified to build MinGW GnuCOBOL much more quickly and easily.

At this point the next step is to build the optional indexed sequential file support (either BDB, no indexed sequential file support, or VBISAM 2.0), before building from GnuCOBOL 2.0 source code files.

## Building Berkeley Database (BDB)

The **Berkeley Database (BDB)** file is named "db-6.2.23.NC.tar.gz", which is the most current version as of September, 2016, and it can be downloaded from:
http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/downloads/index.html

The Berkeley Database (BDB) package provides indexed file access for the GnuCOBOL compiler. You should bypass this step if you intend to build GnuCOBOL with VBISAM 2 instead of BDB, or if you intend to build GnuCOBOL with no Indexed Sequential file support (NODB).

Copy "**db-6.2.23.NC.tar.gz**" into your **C:\GC20-build\MinGW\MSYS** folder and then use the following MSYS bash commands to decompress it:

```
cd /mingw/MSYS
tar xzf db*.tar.gz
rm db*.tar.gz
```

At this point we need to make a **source code patch** to BDB before continuing. Locate the file named "C:\GC20-build\MinGW\MSYS\db-6.2.23.NC\src\os_windows**os_stat.c**" and search for "**_tcsclen**". There should be only one instance of it. Replace "**_tcsclen**" with "**strlen**", and save the file. If you are cautious you may want to first make a backup of that "os_stat.c" file, and then a separate backup of the patched version of that file.

Then build BDB using the following commands:

```
cd db*/build_unix
../dist/configure --enable-mingw --prefix=/mingw --enable-compat185 LIBCSO_LIBS=-lwsock32
make
make install
```

The "configure" command runs fairly quickly and produces hundreds of messages. The "make" command runs for a fairly long time. The "make install" command runs very quickly. Unlike other components the Berkeley Database package does not support "make check" or "make test" to validate the build.

**NOTE: As of 05 December, 2016, only the BDB (and NO-DB) GnuCOBOL versions pass the NIST COBOL85 test suite. The VBISAM 2.0 version has a setup problem that prevents the NIST COBOL85 tests from completing without manual intervention.**

It appeared as if the Berkeley DataBase “make” command failed because there was a long wait after displaying the following messages:

```
libtool: install: cp -p .libs/db_printlog.exe /mingw/bin/db_printlog.exe
libtool: install: cp -p .libs/db_recover.exe /mingw/bin/db_recover.exe
libtool: install: cp -p .libs/db_replicate.exe /mingw/bin/db_replicate.exe
libtool: install: cp -p .libs/db_stat.exe /mingw/bin/db_stat.exe
libtool: install: cp -p .libs/db_tuner.exe /mingw/bin/db_tuner.exe
libtool: install: cp -p .libs/db_upgrade.exe /mingw/bin/db_upgrade.exe
libtool: install: cp -p .libs/db_verify.exe /mingw/bin/db_verify.exe
Installing documentation: /mingw/docs ...
```

If your BDB build appears to hang on this step, simply allow it run for a long time, at least 20 minutes. It should finish eventually. In later builds I found little or no wait for installing documentation.

After completing the “make install” step, run the following command to verify that "**libdb-6.2.dll**" was generated in "**C:\GC20-build\MinGW\bin**":

```
ls /mingw/bin/libdb*.dll
```

If you are using a different version of BDB, the “6.2” in the “libdb” name will match the version number of the BDB package.

As of 01 June 2017, the most current version of BDB from the download site is: "db-6.2.32.NC.tar.gz".

## Building VBISAM 2.0

The alternative Indexed Sequential component is **VBISAM 2.0**, which has a less restrictive license than Oracle Berkeley DataBase. You should bypass this step if you plan to build GnuCOBOL with Oracle Berkeley DataBase (BDB), or NODB (no ISAM support) instead of VBISAM.

VBISAM code is part of the GnuCOBOL open source master which can be downloaded from here:

https://github.com/opensourcecobol/opensource-cobol/archive/master.zip

The zip file will be named opensource-cobol-master.zip. You should unzip this file and then copy the "**vbisam**" subfolder into your "**C:\GC20-build\MinGW\MSYS**" folder using Windows utilities. The "vbisam" folder should already be expanded in "/mingw/MSYS/vbisam", so you do not need to run a "tar" command to decompress it. The modules in the "vbisam" folder are now dated around 08 December 2016, which is newer than my last build.

You may now return to your MSYS bash shell and enter the following commands to build VBISAM:

```
cd /mingw/MSYS
cd vbi*
./configure --prefix=/mingw
make
make check
make install
```

The "configure" command runs fairly quickly and produces about 100 messages. The "make" command runs a bit longer and produces more messages. The "make check" command runs very quickly. The "make install" command should also run very quickly.

There are other sources for VBISAM 2.0, and you should check the GnuCOBOL SourceForge discussion groups to see if a newer version is available, and if the NIST COBOL85 tests will run.

**GnuCOBOL 2.0 Build Guide for MinGW (draft)**

**GnuCOBOL 2.0** Release Candidate 2 source code was downloaded from the following sourceforge link:

https://sourceforge.net/projects/open-cobol/files/gnu-cobol/2.0/

The downloaded file was named "**gnu-cobol-2.0_rc-2.tar.gz**" and dated 2016-11-06. Copy that file into your MSYS folder. Then use the following MSYS bash shell commands to unpack the GnuCOBOL 2.0 Release Candidate 2 source code:

```
cd /mingw/MSYS
tar xzf gnu*.tar.gz
rm gnu*.tar.gz
```

When you are finished you should have a subfolder named "gnu-cobol-2.0" in your **/MinGW/MSYS** folder.

**Applying Patches**

When I built Release Candidate 2, unlike Release Candidate 1, there were no source patches that needed to be applied to the GnuCOBOL files.

At this point, if you are not connected to the internet and you also want to run the NIST COBOL85 test suite, you will need to copy the "**newcob.val**" file into this folder:

C:\GC20-build\MinGW\msys\gnu-cobol-2.0\tests\cobol85

## Build the GnuCOBOL 2.0 compiler

The next step is to build the GnuCOBOL compiler from source code, using the following commands in MSYS, depending upon whether you want to build with BDB, or VBISAM, or without any ISAM support at all (--without-db):

```
cd /mingw/MSYS/gnu*
./configure --prefix=/mingw --disable-rpath
      /* to build with BDB */
make
make check               /* Test the GnuCOBOL build */
make test                /* NIST COBOL 85 tests     */
make install
```

-OR-

```
cd /mingw/MSYS/gnu*
./configure --prefix=/mingw --with-vbisam --disable-rpath
make
make check               /* Test the GnuCOBOL build */
make test                /* NIST COBOL 85 tests     */
make install
```

-OR-

```
cd /mingw/MSYS/gnu*
./configure --prefix=/mingw --without-db --disable-rpath
make
make check               /* Test the GnuCOBOL build */
make test                /* NIST COBOL 85 tests     */
make install
```

The default or typical GnuCOBOL path is to build the compiler with Oracle Berkeley DataBase. The next comments assume that was your choice.

The “./configure” step runs for less than one minute and generates one or two screens of diagnostic or informational messages.

The "make" step runs for about 10 minutes and generates hundreds of messages.

The "make check" command  performs 670 basic tests against the GnuCOBOL 2.0 compiler, and takes 10-15 minutes to run.

"make test" (NIST COBOL85 test suite) generates hundreds of messages and runs for about 10 minutes.

Here is the “make check” test results summary from building GnuCOBOL 2.0 RC-2 with BDB:

```
## ------------- ##
## Test results. ##
## ------------- ##

ERROR: 670 tests were run,
4 failed (3 expected failures).
6 tests were skipped.
## ------------------------- ##
## testsuite.log was created. ##
## ------------------------- ##

Please send `tests/testsuite.log' and all information you think might help:

   To: <open-cobol-list@lists.sourceforge.net>
   Subject: [GnuCOBOL 2.0] testsuite: 559 failed

You may investigate any problem if you feel able to do so, in which
case the test suite provides a good starting point.  Its output may
be found below `tests/testsuite.dir'.

327: ACCEPT OMITTED (SCREEN)                        skipped (run_accept.at:96)
391: SORT: table sort (3)                           skipped (run_misc.at:1990)
423: ON EXCEPTION clause of DISPLAY                 skipped (run_misc.at:4637)
424: EC-SCREEN-LINE-NUMBER and -STARTING-COLUMN     skipped (run_misc.at:4662)
425: LINE/COLUMN 0 exceptions                       skipped (run_misc.at:4703)
437: First READ on empty SEQUENTIAL INDEXED file    skipped (run_file.at:61)
559: Formatted funcs w/ invalid variable format     FAILED (run_functions.at:3690)
584: CALL BY VALUE alphanumeric item                expected failure (run_extensions.at:1022)
595: PROCEDURE DIVISION USING BY ...                expected failure (run_extensions.at:1393)
603: SORT ASSIGN KEYBOARD to ASSIGN DISPLAY         expected failure (run_extensions.at:1796)
```

This was a  successful outcome.  Only 4 tests out of 670 failed with BDB.

Here are the results from running the “make test” step for NIST COBOL85 testing with Berkeley Database:

```
Comparing total test results
diff ./summarynoix.txt summary.log
13c13
< DBNOIX       14       14     0     0    389    0       4      26   419
---
> DBNOIX       14       14     0     0    386    0       4      25   415
15c15
< Total       376      376     0     0   9022    0      19     142  9183
---
> Total       376      376     0     0   9019    0      19     141  9179
make[2]: *** [test] Error 1
make[2]: Leaving directory `/mingw/MSYS/gnu-cobol-2.0/tests/cobol85'
make[1]: *** [test] Error 2
make[1]: Leaving directory `/mingw/MSYS/gnu-cobol-2.0/tests'
make: *** [test-only] Error 2
```

The NIST COBOL85 test suit reported an error at the end of the test, but the “summary.log” file was created in the /tests/COBOL85 folder to report these test results:

```
------ Directory Information -------   --- Total Tests Information ---
Module Programs Executed Error Crash   Pass Fail Deleted Inspect Total
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
NC           95       95     0     0   4371    0       4      26  4401
SM           17       17     0     0    293    0       2       1   296
IC           25       25     0     0    247    0       4       0   251
SQ           85       85     0     0    521    0       0      89   610
RL           35       35     0     0   1830    0       5       0  1835
ST           40       40     0     0    289    0       0       0   289
SG           13       13     0     0    313    0       0       0   313
OB            7        7     0     0     34    0       0       0    34
IF           45       45     0     0    735    0       0       0   735
DBNOIX       14       14     0     0    386    0       4      25   415
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
Total       376      376     0     0   9019    0      19     141  9179
```

This was a very good result, with 9019 tests out of 9179 passing, and no program abends or failed tests.

In Early December, 2016, I tried building Release Candidate 2 with VBISAM 2.0 again. The "make test" for the NIST COBOL85 test suite went into an infinite loop testing the IX209A program. I quickly terminated IX209A using Windows Task Manager after it had recorded 37 million I/O's for its log file. The tests completed, but the results were not as reassuring as the NIST COBOL85 test results for BDB and NO-DB.

First, the "make check" results for Release Candidate 2 with VBISAM were almost as good as BDB:

```
## ------------- ##
## Test results. ##
## ------------- ##

ERROR: 671 tests were run,
5 failed (3 expected failures).
5 tests were skipped.
## -------------------------- ##
## testsuite.log was created. ##
## -------------------------- ##

Please send `tests/testsuite.log' and all information you think might help:

   To: <open-cobol-list@lists.sourceforge.net>
   Subject: [GnuCOBOL 2.0] testsuite: 437 559 failed

327: ACCEPT OMITTED (SCREEN)                         skipped (run_accept.at:96)
391: SORT: table sort (3)                            skipped (run_misc.at:1990)
423: ON EXCEPTION clause of DISPLAY                  skipped (run_misc.at:4637)
424: EC-SCREEN-LINE-NUMBER and -STARTING-COLUMN      skipped (run_misc.at:4662)
425: LINE/COLUMN 0 exceptions                        skipped (run_misc.at:4703)
437: First READ on empty SEQUENTIAL INDEXED file     FAILED (run_file.at:93)
559: Formatted funcs w/ invalid variable format      FAILED (run_functions.at:3690)
584: CALL BY VALUE alphanumeric item                 expected failure
       (run_extensions.at:1022)
595: PROCEDURE DIVISION USING BY ...                 expected failure
       (run_extensions.at:1393)
603: SORT ASSIGN KEYBOARD to ASSIGN DISPLAY          expected failure
       (run_extensions.at:1796)
```

Test 437 (First READ on empty SEQUENTIAL INDEXED file) failed with VBISAM, and it did not fail with Berkeley DataBase.

The Release Candidate 2 “make test” NIST COBOL85 tests with VBISAM 2.0 produced the following summary log:

| ------ Directory Information ------- | | | | | --- Total Tests Information --- | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Module | Programs | Executed | Error | Crash | Pass | Fail | Deleted | Inspect | Total |
| NC | 95 | 95 | 0 | 0 | 4371 | 0 | 4 | 26 | 4401 |
| SM | 17 | 17 | 0 | 0 | 293 | 0 | 2 | 1 | 296 |
| IC | 25 | 25 | 0 | 0 | 247 | 0 | 4 | 0 | 251 |
| SQ | 85 | 85 | 0 | 0 | 521 | 0 | 0 | 89 | 610 |
| RL | 35 | 35 | 0 | 0 | 1830 | 0 | 5 | 0 | 1835 |
| IX | 42 | 6 | 0 | 16 | 61 | 732 | 55 | 0 | 848 |
| ST | 40 | 40 | 0 | 0 | 289 | 0 | 0 | 0 | 289 |
| SG | 13 | 13 | 0 | 0 | 313 | 0 | 0 | 0 | 313 |
| OB | 7 | 7 | 0 | 0 | 34 | 0 | 0 | 0 | 34 |
| IF | 45 | 45 | 0 | 0 | 735 | 0 | 0 | 0 | 735 |
| DB | 15 | 14 | 0 | 1 | 386 | 0 | 4 | 25 | 415 |
| Total | 419 | 382 | 0 | 17 | 9080 | 732 | 74 | 141 | 10027 |

This was not as good a result as BDB. VBISAM passed 9,080 tests out of 10,027, with 732 tests failing. Some users may prefer VBISAM because it has less restrictive licensing terms than Oracle Berkeley DataBase, but as of December, 2016, Berkeley DataBase appears to be more stable and to work better with MinGW GnuCOBOL 2.0.

## Packaging the C:\GnuCOBOL folder

Assuming all the previous steps completed successfully, especially "**make install**", here are the logical steps for building the C:\GnuCOBOL folder containing GnuCOBOL 2.0:

Create the C:\GnuCOBOL folder (mkdir C:\GnuCOBOL)

| | |
|---|---|
| copy C:\GC20-build\MinGW\bin | to C:\GnuCOBOL\bin |
| copy C:\GC20-build\MinGW\share\gnu-cobol\config | to C:\GnuCOBOL\config |
| copy C:\GC20-build\MinGW\share\gnu-cobol\copy | to C:\GnuCOBOL\Copy |
| copy C:\GC20-build\MinGW\lib | to C:\GnuCOBOL\lib |
| copy C:\GC20-build\MinGW\libexec | to C:\GnuCOBOL\libexec |
| copy C:\GC20-build\MinGW\include | to C:\GnuCOBOL\include |

Note that these are logical instructions, not explicit copy command syntax. You can also use Windows Explorer to copy these folders. The important thing is to be sure that all subfolders are copied.

The Windows CMD.EXE commands would look like this, assuming "C:\GnuCOBOL" is the name chosen for your compiler folder, and "C:\GC20-build\MinGW" is the name of the Windows folder where MinGW resides:

```
mkdir C:\GnuCOBOL
xcopy C:\GC20-build\MinGW\bin\*.* c:\GnuCOBOL\bin\ /s /e
del c:\GnuCOBOL\bin\auto*.*
xcopy C:\GC20-build\MinGW\share\gnu-cobol\config\*.* C:\GnuCOBOL\config\ /s /e
xcopy C:\GC20-build\MinGW\share\gnu-cobol\copy\*.* C:\GnuCOBOL\copy\ /s /e
xcopy C:\GC20-build\MinGW\lib\*.* C:\GnuCOBOL\lib\
xcopy C:\GC20-build\MinGW\lib\gcc\*.* C:\GnuCOBOL\lib\gcc\ /s /e
xcopy C:\GC20-build\MinGW\libexec\gcc\*.* C:\GnuCOBOL\libexec\gcc\ /s /e
xcopy C:\GC20-build\MinGW\include C:\GnuCOBOL\include\ /s /e
del c:\GnuCOBOL\include\autosp*.*
xcopy C:\GC20-Build\MinGW\msys\gnu-cobol-2.0\extras\*.* C:\GnuCOBOL\extras\ /s
/e
```

These commands could be built into a .BAT or .CMD file if this step will be done more than once.

Simon Sobisch provided commands for removing unneeded components from the generated compiler to reduce the size of the download file. This ".cmd" file would need to be executed in the "C:\GnuCOBOL" folder:

```
echo strip out unneeded GnuCOBOL components
echo.

PAUSE

copy bin\strip* . && copy bin\libiconv* . && strip -p --strip-debug --strip-
unneeded bin\*.dll bin\*.exe lib\*.a && del strip* libiconv*
```

Note that the last two lines are a single statement that concatenates four separate commands.

The resulting C:\GnuCOBOL folder is sufficient for compiling COBOL programs, but some additional files should also be added.

**GnuCOBOL 2.0 Build Guide for MinGW (draft)**

The following files were copied from the MinGW GnuCOBOL 1.1 compiler included in the Windows OpenCOBOLIDE install package, or from other sources:

AUTHORS.txt
ChangeLog.txt
ChangeLog_cobc.txt
ChangeLog_libcob.txt
config.log Copied from C:\GC20-build\MinGW\MSYS\gnu-cobol-2.0\config.log
COPYING.txt
COPYING_DOC.txt
COPYING_LESSER.txt
GnuCOBOL 2.0 Manual.pdf (from the GnuCOBOL Sourceforge website)
NEWS.txt
README.txt Modified for this package
set_env.cmd Modified for this package (written by Simon Sobisch)
testsuite.log Copied from C:\GC20-build\MinGW\MSYS\gnu-cobol-2.0\tests\testsuite.log
THANKS.txt

The "config.log" and "testsuite.log" files are copied from the MinGW MSYS build of GnuCOBOL 2.0, and used to diagnose compiler build problems.

The COPYING, AUTHORS, NEWS, README, and THANKS files can also be downloaded from the GnuCOBOL sources on the project page:

https://sourceforge.net/p/open-cobol/code/HEAD/tree/branches/gnu-cobol-2.0/

Then there are several files I added for quickly testing GnuCOBOL compiler installation:

| | |
|---|---|
| bintest.cbl | Sample COBOL program demonstrating binary number usages |
| gcx.cmd | Command file to compile a .cob or .cbl source file and create .exe and .lst |
| gcmf.cmd | Command file to compile a .cob or .cbl source file and create .dll and .lst<br>The input COBOL source file is assumed to be free-form. |
| testfunc.cob | Sample COBOL program to display date compiled and current date |
| TestGC.cmd | Command file to compile and execute testfunc.cob, once as .exe and once as .dll, and delete the generated exe, dll, and lst files. |

Then the C:\GnuCOBOL" folder was packaged as an “7z” self-extracting file. The self-extracting file was packaged in a zip file and can be downloaded from:

http://www.arnoldtrembley.com/GC20RC2-BDB-7z-sfx.zip

The “NODB” version of GnuCOBOL 2.0 is packaged in a similar manner and can be downloaded from here
http://www.arnoldtrembley.com/GC20RC2-NODB-7z-sfx.zip

The zip file contains instructions for running the self-extracting archive.

Based on recommendations from Simon Sobisch, I have changed the packaging of the compiler to use open source 7-Zip self-extracting archives which provide better compression ratios. Due to a security restriction from my web hosting service I cannot host “.exe” files. So the new files have been renamed with “.7z” as their file extension. After downloading they can be opened using 7-Zip, or the windows file extensions can be renamed from “.7z” to “.exe”, allowing them to be used as self-extracting archives.

7-Zip is open source software available from http://www.7-zip.org/

As of December 5, 2016, the newest MinGW binaries for GnuCOBOL 2.0 Release Candidate 2 can be downloaded from the following addresses:

http://www.arnoldtrembley.com/GC20RC2-BDB-rename-7z-to-exe.7z

http://www.arnoldtrembley.com/GC20RC2-NODB-rename-7z-to-exe.7z

http://www.arnoldtrembley.com/GC20RC2-VBI-rename-7z-to-exe.7z

In the future, new binaries will be added to the following page:

http://www.arnoldtrembley.com/GnuCOBOL.htm

OpenCOBOLIDE is a GUI (Graphical User Inteface) written in Python and used to edit, compile, and test GnuCOBOL programs. It is compatible with Unix/Linux, Windows, and Mac OSX. The OpenCOBOLIDE preferences can be changed to use the extracted GnuCOBOL 2.0 RC2 compiler instead of the GnuCOBOL 1.1 compiler embedded in the Windows installer for OpenCOBOLIDE.

The OpenCOBOLIDE install package for Windows can be downloaded from this site:
https://launchpad.net/cobcide/4.0/4.7.6/+download/OpenCobolIDE-4.7.6_Setup.exe

Additional information about OpenCOBOLIDE can be found here:
http://opencobolide.readthedocs.io/en/latest/download.html

The "GnuCOBOL 2.0 Manual.pdf" was downloaded from the GnuCOBOL sourceforge website:
https://open-cobol.sourceforge.io/doc/gnucobol.pdf

The GnuCOBOL 2.0 Programmer's guide (640 pages) which can be downloaded from:
https://open-cobol.sourceforge.io/guides/GNU%20COBOL%202.0%20Programmers%20Guide.pdf

The parent page for all GnuCOBOL manuals and related documentation downloads is:
https://open-cobol.sourceforge.io/

**Future Steps**

As of November 13, 2016, I have successfully run the NIST COBOL85 test suite with GnuCOBOL 2.0 with BDB and with “NODB”. In early December I was able to run the NIST COBOL85 test suite with VBISAM 2.0, but manual intervention is required and the results are not as clean. At some point the VBISAM 2.0 code and setup for MinGW, CygWIN, and native Windows will hopefully be fixed.

I plan to test building GnuCOBOL with MinGW built-in gmplib and curses support, which would allow this document to be simplified.

I also plan to add instructions for building a normal Windows “setup.exe” file using INNO.

The free INNO installer program can be found at this site:
http://www.jrsoftware.org/isinfo.php