GnuCOBOL 3.1.1 [08DEC2020 - Release]
Build Guide for MinGW - Minimalist GNU for Windows

cobc (GnuCOBOL) 3.1.1

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 Wednesday, 09 December 2020.

My original starting point for building GnuCOBOL and writing these guides was the manual written by Gary Cutler (CutlerGL@gmail.com).

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

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

Simon Sobisch of the GnuCOBOL project 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 project space can be found here:

https://www.gnu.org/software/gnucobol

https://sourceforge.net/projects/gnucobol/

GnuCOBOL 3.1.1 Build Guide for MinGW (draft)

Required Components:

You will need to download the following components in order to build the GnuCOBOL 3.1.1 compiler in Windows:

1. MinGW - Minimalist GNU for Windows
2. GNU Multiple-Precision Arithmetic package (gmplib)
3. PDCursesMod 4.2.0 - used for screen read and write. (alternate versions available)
4. Berkeley Database (BDB) package from Oracle ==OR==
5. VBISAM2 by Roger While
6. GnuCOBOL 3.1.1 compiler source code

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

Licensing:

The GnuCOBOL 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 included (NODB).

Download the packages:

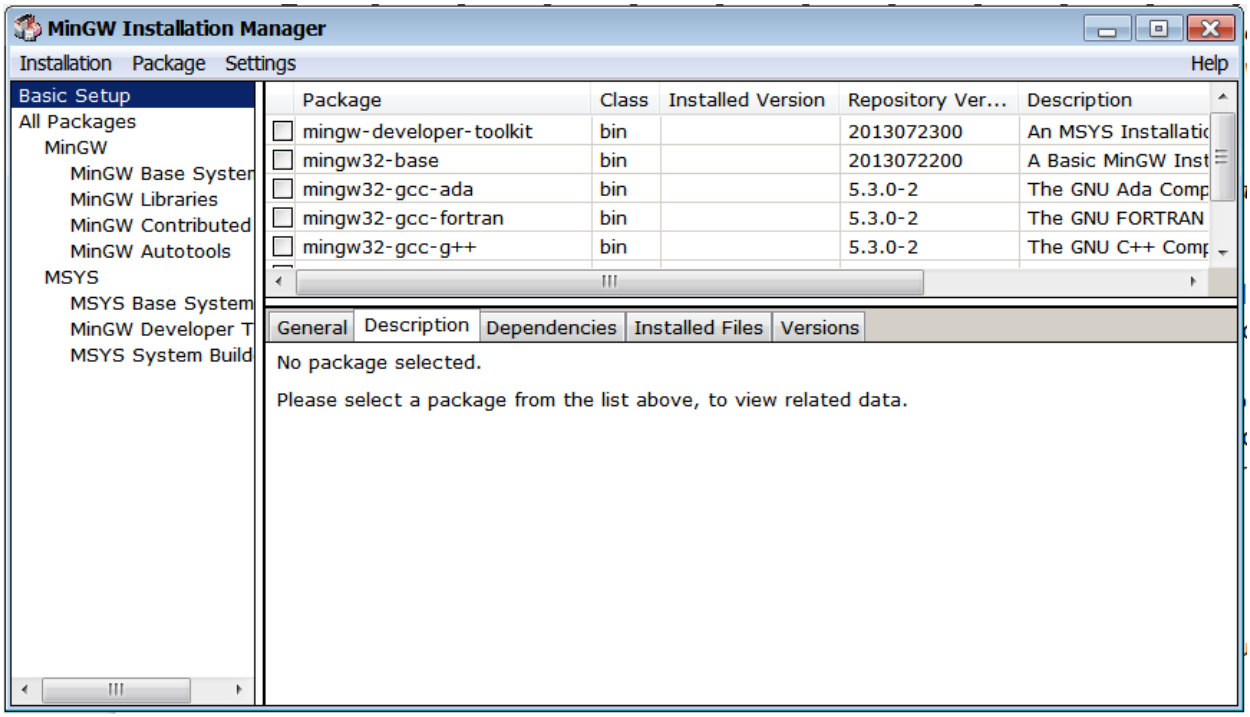
The **MinGW** software package can be downloaded from:
https://sourceforge.net/projects/mingw/files/
https://sourceforge.net/projects/mingw/files/Installer/

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. It is only needed to build the compiler. The generated GnuCOBOL compiler will run in a Windows CMD.EXE shell. GnuCOBOL will translate COBOL source code into C source code, and will call the embedded MinGW GCC compiler to compile the intermediate C code into an executable program for Windows.

Installing MinGW and the MSYS bash shell

When you run “mingw-get-setup.exe” it will default to installing in **C:\MinGW** and you should accept this default. But when you are using any generated GnuCOBOL compiler you should **rename this folder**, to avoid path/name conflicts with the MinGW GCC compiler embedded in GnuCOBOL.

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. Some of them will already be selected with **mingw32-base** and **msys-base**.

```
mingw32-bzip2.bin
mingw32-dos2unix.bin
mingw32-gettext.bin         /* for translatable strings */
mingw32-gettext.dev         /* for translatable strings */
mingw32-gettext.dll         /* for translatable strings */
mingw32-libtool.bin
mingw32-make.bin
msys-bash.bin               /* to include the MinGW bash shell */
msys-m4.bin                 /* for building Berkeley DB */   already in...
msys-make.bin                       already in...
msys-perl.bin               /* for NIST COBOL85 testing */   dev toolkit
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.

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

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

```
c:/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:\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:/MinGW                          /mingw
```

MSYS will only use the “/mingw” mount point, even though it is "**C:\MinGW**" to Windows (and indeed, could be a completely different name in Windows). If you selected a different device and folder name on the MinGW installation 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:\MinGW\MSYS\1.0\msys.bat -norxvt
```

Do not forget to add 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:\MinGW\MSYS\1.0\msys.ico”.

Make sure the properties for this shortcut use:

Target: C:\MinGw\msys\1.0\msys.bat -norxvt
Start in: C:\MinGW\MSYS\1.0

Also, make sure the Security properties of the MSYS shortcut include **FULL administrator access** (Full Control, Modify, Read & Execute, Read, and Write).

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

How to Configure the BASH shell (*From Gary Cutler***):**

The window started by the "MSYS" icon resembles a Windows console, but is actually an MSYS "bash" shell for running the MinGW GCC development environment. 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 4000 lines. When running "make" commands many components display thousands of messages in the MSYS window, and you may want to scroll back to view them. You will also want to change the font to something that enables that window to fit on your screen, for example, "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**".

If you are running MinGW on a widescreen laptop you may want to limit the window height to only 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 (by highlighting with the mouse) 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 GnuCOBOL discussion forums, so your posts can appear immediately without having to wait to be moderated.

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

https://sourceforge.net/projects/gnucobol/files/nist/

https://sourceforge.net/p/gnucobol/discussion/

In previous builds some of the errors I encountered included messages like this:

```
rm: cannot lstat `conftest.exe': Permission denied
```

Based on comments in the GnuCOBOL forums, the best solution to this problem is to disable Windows file indexing (for search), and also to deactivate your internet security/anti-virus product temporarily. But you can safely leave Windows Defender/Microsoft Security Essentials (MSE) activated.

Anti-Virus Considerations

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

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:\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.

Some of the errors I encountered said "permission denied", and when I reran that configure or make step, the errors disappeared.

I recommend running with your anti-virus disabled (except for Windows Defender/MSE). This will allow the build to run much more quickly and also prevent problems during the build.

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

https://sourceforge.net/projects/gnucobol/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 6.2.0** GNU Multiple-Precision Arithmetic Library can be downloaded from:
https://gmplib.org/#DOWNLOAD
https://gmplib.org/download/gmp/gmp-6.2.0.tar.xz

As of 17 January 2020, the most current GMP version is in a file named "gmp-6.2.0.tar.lz", but you can search the site to find a download in tar.xz format, which is easier to unpack in MinGW.

https://gmplib.org/download/gmp/gmp-6.2.0.tar.xz

Copy "gmp-6.2.0.tar.xz" into your **C:\MinGW\MSYS** folder using Windows commands, and then use the following bash shell commands to uncompress it:

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

At this point you have a Windows folder named C:\MinGW\MSYS\gmp-6.2.0. The next five MSYS commands will build gmp 6.2.0 for GnuCOBOL:

```
cd gmp*
./configure -prefix=/mingw --enable-fat --enable-shared --disable-static
make
make check          # Test gmp 6.2.0
make install

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

You may find it easier to copy and paste these commands into the MSYS bash shell one line at at time, especially for the "./configure" command. 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 about 5 minutes to run and produced hundreds of messages.

The "make" command took about 15 minutes to run and produced thousands of messages.

The "make check" command ran for about 15 minutes 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 WITH PDCURSESMOD 4.2.0 (WINCON)

The PDCurses package is used for COBOL SCREEN-SECTION and extended console input-output support (DISPLAY/ACCEPT AT/WITH).

Download PDCursesMod 4.2.0, which can be found at the following location:

https://github.com/Bill-Gray/PDCursesMod/archive/master.zip

https://github.com/Bill-Gray/PDCursesMod/archive/v4.2.0.tar.gz

Copy "PDCursesMod-4.2.0.tar.gz" into your **C:\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
```

After you have unpacked the tarball, you would use the following commands to build PDCurses:

```
cd PDC*/wincon      # note changes
make -f Makefile INFOEX=N CHTYPE_64=Y DLL=Y
```

Then use the following command to verify that a "pdcurses.dll" exists in the "**C:\MinGW\MSYS\PDCursesMod-4.2.0\wincon**" folder.

```
ls *.dll
```

You can also verify the presence of “pdcurses.dll” using Windows Explorer.

There is no “make install” in PDCurses , so you must enter the following MSYS commands into the bash shell to install PDCurses 4.1.0 (either wincon or wingui). This is another instance where it may help you to paste these commands into the MSYS bash shell window:

```
install pdcurses.dll /mingw/bin/.
install pdcurses.a /mingw/lib/libpdcurses.a
cd ..        (NOTE!!!)
install *.h /mingw/include/.
install curses.h /mingw/include/pdcurses.h
```

GnuCOBOL 3.1.1 Build Guide for MinGW (draft)

NOTE!!!

Before installing **curses.h** into **pdcurses.h**, edit it as suggested by Simon Sobisch:

* copy curses.h to pdcurses.h (in the same folder)
* insert the necessary defines into pdcurses.h (see example below)
* install pdcurses.h

The change to apply _in this case_:

```diff
 #ifndef __PDCURSES__
 #define __PDCURSES__ 1
```

Immediately after the lines above, and the lines found here:

```
/* defines matching the creation of the library */
 #define CHTYPE_64
#define PDC_DLL_BUILD
```

In other cases you'd include `#define CHTYPE_32`, `#define PDC_WIDE`, `#define PDC_FORCE_UTF8` there, too.

This way you will always be able to determine the actual pdcurses defines used.

```
install pdcurses.h /mingw/include/pdcurses.h
```

Building PDCursesMod 4.2.0 (for WINGUI)

If you want to build PDCurses 4.2.0/**wingui** (instead of "**wincon**" version), here are instructions that should work. But be aware that "stderr" will be directed to a cmd.exe window separate from the current screen which may not be desireable:

Copy "PDCursesMod-4.2.0.tar.gz" into your **C:\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 PDCursesMod 4.2.0/**wingui** using these commands:

```
cd PDC*/wingui      # note changes
make -f Makefile INFOEX=N CHTYPE_32=Y DLL=Y WIDE=Y
```

Then use the following command to verify that a "pdcurses.dll" exists in the "**C:\MinGW\MSYS\PDCursesMod-4.2.0\wingui**" folder.

```
ls *.dll
```

There is no "make install" in PDCurses , so you must enter the following MSYS commands into the bash shell to install PDCurses 4.1.0 (either wincon or wingui). This is another instance where it may help you to paste these commands into the MSYS bash shell window:

```
install pdcurses.dll /mingw/bin/.
install pdcurses.a /mingw/lib/libpdcurses.a
cd ..
install *.h /mingw/include/.
install curses.h /mingw/include/pdcurses.h (SEE NOTE ABOVE)
```

This page is notes for alternate PDCurses 4.2.0 build options, or things to be careful with...

(use the following for wincon or wingui:)

```
make -f Makefile INFOEX=N CHTYPE_32=Y DLL=Y WIDE=Y UTF8=Y
```

Some users may prefer to build GnuCOBOL with PDCurses 3.9 because the colors look “better” than with PDCurses 4.1.0 (either Wincon of WinGUI), but the PDCurses color problems have been resolved with 4.2.0 (although the build is more complicated). PDCurses 4.2.0 also has more extensive mouse support than version 3.9.

BUILDING WITH PDCURSES 3.9

The PDCurses package is used for COBOL SCREEN-SECTION and extended console input-output support (DISPLAY/ACCEPT AT/WITH). You can download PDCurses 3.9 from here:

https://github.com/wmcbrine/PDCurses/releases

https://github.com/wmcbrine/PDCurses/archive/3.9.tar.gz

Copy "PDCurses-3.9.tar.gz" into your **C:\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
```

After you have unpacked the tarball, you would use the following commands to build PDCurses:

```
cd PDC*/wincon
make -f Makefile INFOEX=N CHTYPE_32=Y DLL=Y
```

Then use the following command to verify that a "pdcurses.dll" exists in the "**C:\MinGW\MSYS\PDCurses-3.9\wincon**" folder.

```
ls *.dll
```

You can also verify the presence of “pdcurses.dll” using Windows Explorer.

There is no “make install” in PDCurses , so you must enter the following MSYS commands into the bash shell to install PDCurses 3.4. These install commands are the same for both PDCurses 3.4 and PDCurses 4.1.0. This is another instance where it may help you to paste these commands into the MSYS bash shell window:

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

Process Improvement Checkpoint

At this point in the build process I normally backup the entire "C:\MinGW" 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, VBISAM 2.0.1,or no indexed sequential file support (NODB). The backup folder includes the MinGW get program and the shortcut to MSYS:

```
 Directory of C:\MinGW-bkup3

12/06/2019  12:08 AM    <DIR>          .
12/06/2019  12:08 AM    <DIR>          ..
12/06/2019  12:09 AM    <DIR>          MinGW
12/05/2019  11:09 PM             2,309 GCBUILD.lnk
09/07/2014  01:53 AM            86,528 mingw-get-setup.exe
10/28/2016  11:35 PM             2,158 msys.bat - Shortcut.lnk
```

For example, if I have already built GnuCOBOL with BDB and I want to build another version with VBISAM 2.0.1, I simply restore the C:\MinGW 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:\MinGW 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.

At this point the next step is to build the optional indexed sequential file support (either BDB, no indexed sequential file support, or VBISAM 2.0.1), before building from GnuCOBOL 3.1.1 source code files.

Building Berkeley Database (BDB)

The **Berkeley Database (BDB)** file is named "db-18.1.40.tar.gz", and it can be downloaded from:

https://download.oracle.com/berkeley-db/db-18.1.40.tar.gz

https://www.oracle.com/database/technologies/related/berkeleydb-release-history.html

Note that registration is required to download BDB. 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.x instead of BDB, or if you intend to build GnuCOBOL with NO Indexed Sequential file support (NODB).

Copy "**db-18.1.40.tar.gz**" into your **C:\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:\MinGW\MSYS\db-18.1.40\src\os_windows**os_stat.c**" and search for "**strlen**". There should be only one instance of it. Replace "**strlen**" 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 --with-
mutex=x86/gcc-assembly --with-cryptography=no
make
make install
```

It is very important to include the "**--with-cryptography=no**" parameter if you wish to export the GnuCOBOL compiler or GnuCOBOL compiled programs outside the USA.

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.

On at least one occasion 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. The delay was probably caused by the Anti-Virus program analyzing all the doc files being copied.

After completing the “make install” step, run the following command to verify that "**libdb-18.1.dll**" was generated in "**C:\MinGW\bin**":

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

If you are using a different version of BDB, the “18.1” in the “libdb” name will match the version number of the BDB package.

As of 28 October 2020, the most current version of BDB from the download site is: "db-18.1.40.tar.gz".

Building VBISAM 2.0.1

The alternative Indexed Sequential component is **VBISAM 2.0.1**, 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.

NOTE: The best version of VBISAM that I found is “vbisam-2.0.1”, posted anonymously in the GnuCOBOL discussion forums. I am hosting it on my website in two different archives (it’s the same source code):

https://www.arnoldtrembley.com/vbisam_install_guide_v5.1.zip

https://www.arnoldtrembley.com/vbisam-2.0.1.zip

You can use either the “vbisam-2.0.1.zip” from the install guide package, or the “vbisam-2.0.1.zip” file by itself. Simply extract “vbisam-2.0.1.zip” into your MSYS folder as a “vbisam-2.0.1” folder. Most of the source modules have a 2016-04-27 05:01:50 date-time stamp.

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

```
cd /mingw/MSYS/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.

After completing the “make install” step, run the following command to verify that "**libvbisam-1.dll**" was generated in "**C:\MinGW\bin**":

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

The vbisam 2.0.1 version used for this build was posted anonymously in the GnuCOBOL forums , and can be downloaded from here:

https://sourceforge.net/p/gnucobol/discussion/help/thread/51d48303/5d39/attachment/vbisam_install_guide_v4.7z

This version of vbisam is based on VBISAM 2.0 by Roger While, found at this address:

https://sourceforge.net/projects/vbisam/files/vbisam2/

But it has patches from Sergey Kashyrin at www.kiska.net, and some additional edits by Mario Matos, and some changes by the anonymous poster.

A newer version of VBISAM (2.2?) should be available by the end of 2020, but I do not yet know where it will be hosted.

Build the GnuCOBOL 3.1.1 compiler

GnuCOBOL 3.1.1 compiler source code was downloaded from the following link:

https://sourceforge.net/projects/gnucobol/files/gnucobol/3.1/

The downloaded file was named "**gnucobol-3.1.1.tar.xz**" and dated 2020-12-08. It is also available in zip, 7z, and tar.gz formats. Copy that file into your MSYS folder. Then use the following MSYS bash shell commands to unpack the GnuCOBOL source code:

```
cd /mingw/MSYS
tar xf gnu*.tar.xz      (or tar xzf gnu*.tar.gz)
rm gnu*.tar.xz          (or rm gnu*.tar.gz)
```

When you are finished you should have a subfolder named "gnucobol-3.1.1" in your **/MinGW/MSYS** folder.

If you are using a “tar.gz” file instead of a “tar.gz” file the unpack command is “tar xzf gnu*.tar.gz” and the delete command is “rm gnu*.tar.gz”.

To enable JSON support in GnuCOBOL, install the cJSON.c and cJSON.h files into \GnuCOBOL\ libcob. You can do this by executing the following commands in the bash shell:

```
cd /mingw/MSYS/gnu*/libcob
wget https://raw.githubusercontent.com/DaveGamble/cJSON/master/cJSON.h
wget https://raw.githubusercontent.com/DaveGamble/cJSON/master/cJSON.c
cd /mingw/MSYS/gnu*
```

You can also download those two files ahead of time from the following locations:

```
https://github.com/DaveGamble/cJSON/blob/master/cJSON.c
https://github.com/DaveGamble/cJSON/blob/master/cJSON.h
```

Or you can download the full cJSON package from GitHub and extract cJSON.c and cJSON.h:

https://github.com/DaveGamble/cJSON/archive/master.zip

If you have disconnected from the internet and you also want to run the NIST COBOL85 test suite, you should copy your previously downloaded "**newcob.val**" file into this folder:

C:\MinGW\msys\gnucobol-3.1.1\tests\cobol85

This may be also be more safe if you have disabled your Windows anti-virus program.

The "newcob.val" file can be downloaded in several different compressed formats from here:

https://sourceforge.net/projects/gnucobol/files/nist/

GnuCOBOL 3.1.1 compiler – configure and make

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 [CPPFLAGS] --prefix=/mingw --with-db --disable-rpath
      /* to build with BDB */

-OR-

./configure [CPPFLAGS] --prefix=/mingw --with-vbisam --disable-rpath
     /* to build with VBISAM 2.0.1 */

-OR-

./configure [CPPFLAGS] --prefix=/mingw --without-db --disable-rpath
     /* to build without Indexed Sequential access (NODB) */
```

If you plan to build the GnuCOBOL compiler so that its internal workings can be debugged, add the following two parameters to your ./configure statement:

```
--enable-debug --enable-cobc-internal-checks
```

After running ./configure, here are the next four steps to run:

```
make                    # Build the GnuCOBOL compiler
make check              # Test the GnuCOBOL build
make test               # Run NIST COBOL 85 test suite
make install            # Prepare Install format
```

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

Here is a sample configure statement with the default **CPPFLAGS** parameter, which as you see must be enclosed in quotes.

```
./configure "CPPFLAGS=-DCHTYPE_64 -DPDC_DLL_BUILD -Dinitscr=initscr_x64" --
prefix=/mingw --with-db --disable-rpath --enable-debug --enable-cobc-internal-
checks
```

If PDCurses is built with UTF support, the last part of CPPFLAGS should be:
-Dinitscr=initscr_u64 (or u32 for CHTYPE_32)
use initscr_w64 (or w32) for wide characters without UTF8 support.

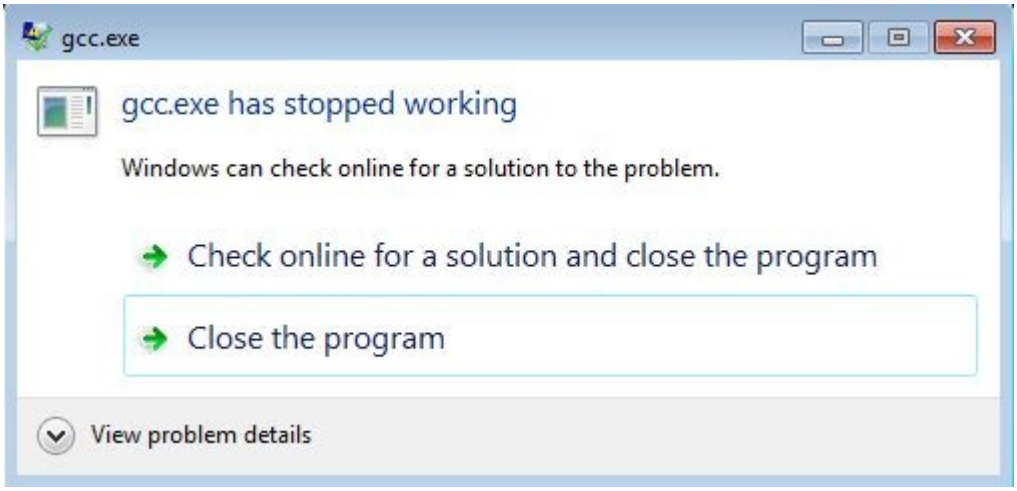
The "./configure" step runs for less than one minute and generates one or two screens of diagnostic or informational messages.

The "make" step runs for about 2 minutes and generates hundreds of messages.

The "make check" step performs 1,036 basic tests against the GnuCOBOL 3.1.1 compiler, and takes 10-15 minutes to run.

The "make test" (NIST COBOL85 test suite) generates hundreds of messages and runs for about 5-10 minutes. "make install" should run in a minute or less.

Test #8 Temporary path test failed – had to close GCC for that one test. Windows popup said GCC failed.



Also test #964 "initialize occurs unbounded" failed in a similar way

Simon Sobisch suggested some alternate versions of the make commands. You can use these if your build PC has 4 CPU's (or 4 threads), and they should speed up the long-running "make" steps.

```
make -j4
make check TESTSUITEFLAGS="--jobs=4"
make -j4 test
```

I have not tested these alternate "make" commands. My build machine is an eight year old laptop with Windows 7 PRO 64-bit and an SSD. The processor is an Intel Sandy Bridge Core i7, 2.80 Ghz, with 8 gigabytes of RAM.

Here are the “make check” test results summary from building GnuCOBOL with COBOL ReportWriter and BDB for ISAM:

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

ERROR: 1066 tests were run,
35 failed (29 expected failures).
17 tests were skipped.
## -------------------------- ##
## testsuite.log was created. ##
## -------------------------- ##

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

   To: <bug-gnucobol@gnu.org>
   Subject: [GnuCOBOL 3.1.1] testsuite: 8 19 20 21 23 761 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'.
```

This was a very successful outcome.

Here are the “make check” test results summary from building GnuCOBOL with COBOL ReportWriter and no ISAM support (NODB):

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

ERROR: 1040 tests were run,
28 failed (22 expected failures).
43 tests were skipped.
## ------------------------- ##
## testsuite.log was created. ##
## ------------------------- ##

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

   To: <bug-gnucobol@gnu.org>
   Subject: [GnuCOBOL 3.1.1] testsuite: 8 19 20 21 23 761 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'.
```

This was a very successful outcome.

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

```
------ Directory Information -------   --- Total Tests Information ---
Module Programs Executed Error Crash   Pass Fail Deleted Inspect Total
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
NC           95       93     0     0   4388    5       4       1  4398
SM           17       17     0     0    291    0       2       1   294
IC           25       25     0     0    246    0       4       0   250
SQ           85       85     0     0    518    0       0      89   607
RL           35       35     0     0   1827    0       5       0  1832
ST           40       40     0     0    288    0       0       0   288
SG           13       13     0     0    310    0       0       0   310
OB            7        7     0     0     39    0       0       0    39
IF           45       45     0     0    733    0       0       0   733
RW            6        6     0     0     40    0       0       0    40
DB           16       16     0     0    418    0       4      27   449
IX           42       42     0     0    507    0       1       0   508
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
Total       426      424     0     0   9605    5      20     118  9748
```

This was a very good result, with 9605 tests passing out of 9748, and no program abends or failed tests.

In previous builds I also tested GnuCOBOL with a small COBOL source file in OpenCobolIDE 4.7.6 (by changing the OpenCobolIDE preferences) and no problems appeared.

Here are the NIST COBOL85 test results for GnuCOBOL with no ISAM support (NODB):

```
------ Directory Information -------   --- Total Tests Information ---
Module Programs Executed Error Crash   Pass Fail Deleted Inspect Total
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
NC           95       93     0     0   4388    5       4       1  4398
SM           17       17     0     0    291    0       2       1   294
IC           25       25     0     0    246    0       4       0   250
SQ           85       85     0     0    518    0       0      89   607
RL           35       35     0     0   1827    0       5       0  1832
ST           40       40     0     0    288    0       0       0   288
SG           13       13     0     0    310    0       0       0   310
OB            7        7     0     0     39    0       0       0    39
IF           45       45     0     0    733    0       0       0   733
RW            6        6     0     0     40    0       0       0    40
DBNOIX       15       15     0     0    400    0       4      25   429
------ -------- -------- ----- -----  ----- ---- ------- ------- -----
Total       383      381     0     0   9080    5      19     116  9220
```

This was a very good result, with 9080 tests out of 9220 passing, with no program abends or failed tests.

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.

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

copy C:\MinGW\bin	to C:\GnuCOBOL\bin
copy C:\MinGW\share\gnu-cobol\config	to C:\GnuCOBOL\config
copy C:\MinGW\share\gnu-cobol\copy	to C:\GnuCOBOL\Copy
copy C:\MinGW\lib	to C:\GnuCOBOL\lib
copy C:\MinGW\libexec	to C:\GnuCOBOL\libexec
copy C:\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:\MinGW" is the name of the MinGW build folder in Windows:

```
mkdir C:\GnuCOBOL
xcopy C:\MinGW\bin\*.* c:\GnuCOBOL\bin\ /s /e
del c:\GnuCOBOL\bin\auto*.*
xcopy C:\MinGW\share\gnucobol\config\*.* C:\GnuCOBOL\config\ /s /e
xcopy C:\MinGW\share\gnucobol\copy\*.* C:\GnuCOBOL\copy\ /s /e
xcopy C:\MinGW\lib\*.* C:\GnuCOBOL\lib\
xcopy C:\MinGW\lib\gcc\*.* C:\GnuCOBOL\lib\gcc\ /s /e
xcopy C:\MinGW\libexec\gcc\*.* C:\GnuCOBOL\libexec\gcc\ /s /e
xcopy C:\MinGW\include C:\GnuCOBOL\include\ /s /e
del c:\GnuCOBOL\include\autosp*.*
del c:\GnuCOBOL\include\curspriv.h
xcopy C:\MinGW\msys\gnucobol-3.1.1\extras\*.* C:\GnuCOBOL\extras\ /s /e
```

Note the in the final "xcopy" command, the name "gnucobol-3.1.1" may vary depending upon the actual name of the folder in MSYS after unpacking GnuCOBOL source code.

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.

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

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

GnuCOBOL 3.1.1 Build Guide for MinGW (draft)

The following files can be copied from the repository found at:

https://sourceforge.net/p/gnucobol/code/HEAD/tree/branches/gnucobol-3.x/

AUTHORS.txt
ChangeLog.txt
ChangeLog_cobc.txt
ChangeLog_libcob.txt
COPYING.txt
COPYING_DOC.txt
COPYING_LESSER.txt
GnuCOBOL 3.1.1 Manual.pdf
copied from C:\MinGW\msys\gnucobol-3.1.1\doc\gnucobol.pdf
NEWS.txt

README.txt	Modified for this package
set_env.cmd	Modified for this package (written by Simon Sobisch)

THANKS.txt

The "config.log", “summary.log”, and "testsuite.log" files should be copied from the MinGW MSYS build of GnuCOBOL. They are used to diagnose compiler build problems.

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

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 “gcmf.cmd” assumes the input COBOL source file is in free-format instead of fixed-format 80-byte lines.
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 temporary exe, dll, and lst files.

In actual practice, I created a full folder with all the added files, then made a copy of it before running the “strip” commands supplied by Simon Sobisch.

Then the smaller “C:\GnuCOBOL" folder was packaged as a self-extracting file.

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. The self-extracting file will prompt you to supply a folder name for the compiler. It can also be installed to a drive other than your C: drive.

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

As of December 15, 2020, the newest MinGW binaries for GnuCOBOL 3.1.1 (08Dec2020) can be downloaded from the following addresses:

https://www.arnoldtrembley.com/GC311-BDB-rename-7z-to-exe.7z

https://www.arnoldtrembley.com/GC311-VBI-rename-7z-to-exe.7z

https://www.arnoldtrembley.com/GC311-NODB-rename-7z-to-exe.7z

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

https://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 a different GnuCOBOL compiler instead of the GnuCOBOL 2.0 RC2 (BDB) 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

OpenCobolIDE development ended on 15 October 2017, as announced by the author, and there will be no future versions.

A better option might be the Open Source **VS-Codium**, which is also the basis for Microsoft's commercial **Visual Studio**.

VS-Codium can be downloaded from:
https://github.com/VSCodium/vscodium/releases
look for a release named something like:
VSCodiumUserSetup-ia32-1.50.1.exe (for 32-bit) OR
VSCodiumUserSetup-x64-1.50.1.exe

You can download COBOL syntax coloring/editing rules for VS-Codium or MS Visual Studio from:
https://open-vsx.org/extension/bitlang/cobol

The GnuCOBOL Programmers Guide, Quick Reference Manual, and Sample Programs Manual can be downloaded in either A4 and USA Letter size PDF's from here:
https://sourceforge.net/p/gnucobol/code/HEAD/tree/external-doc/guide/PDFs/

The parent page for GnuCOBOL manuals and related documentation downloads is:
https://gnucobol.sourceforge.io/