Embarcadero Dev-C++: Successfully Modernizing A Popular Windows C++ IDE

December 3, 2020

Eli M., Embarcadero MVP

Overview

Dev-C++[1] is an open-source C++ IDE, first launched by Colin Laplace with Bloodshed Software[2] at the dawn of the millennium. Originally built in Delphi versions 6 and 7, it achieved notable success and with over 67 million downloads. Johan Mes took up the mantle of upgrading the project in 2011 with a new fork called Orwell Dev-C++ but stopped updating in 2015[3]. Embarcadero sponsored the project under the new Embarcadero Dev-C++ fork and upgraded it with modern Windows 10 and C++17/C++20 support.

There are a number of reasons to upgrade the codebase from Delphi 7 to the latest version of Delphi. These include updated High-DPI support, adding styles for light and dark themes to match the editor interface, support for multiple file encodings including Unicode, and modernizing the interface. This is on top of adding things like C++17 and C++20 support through new GCC compilers.

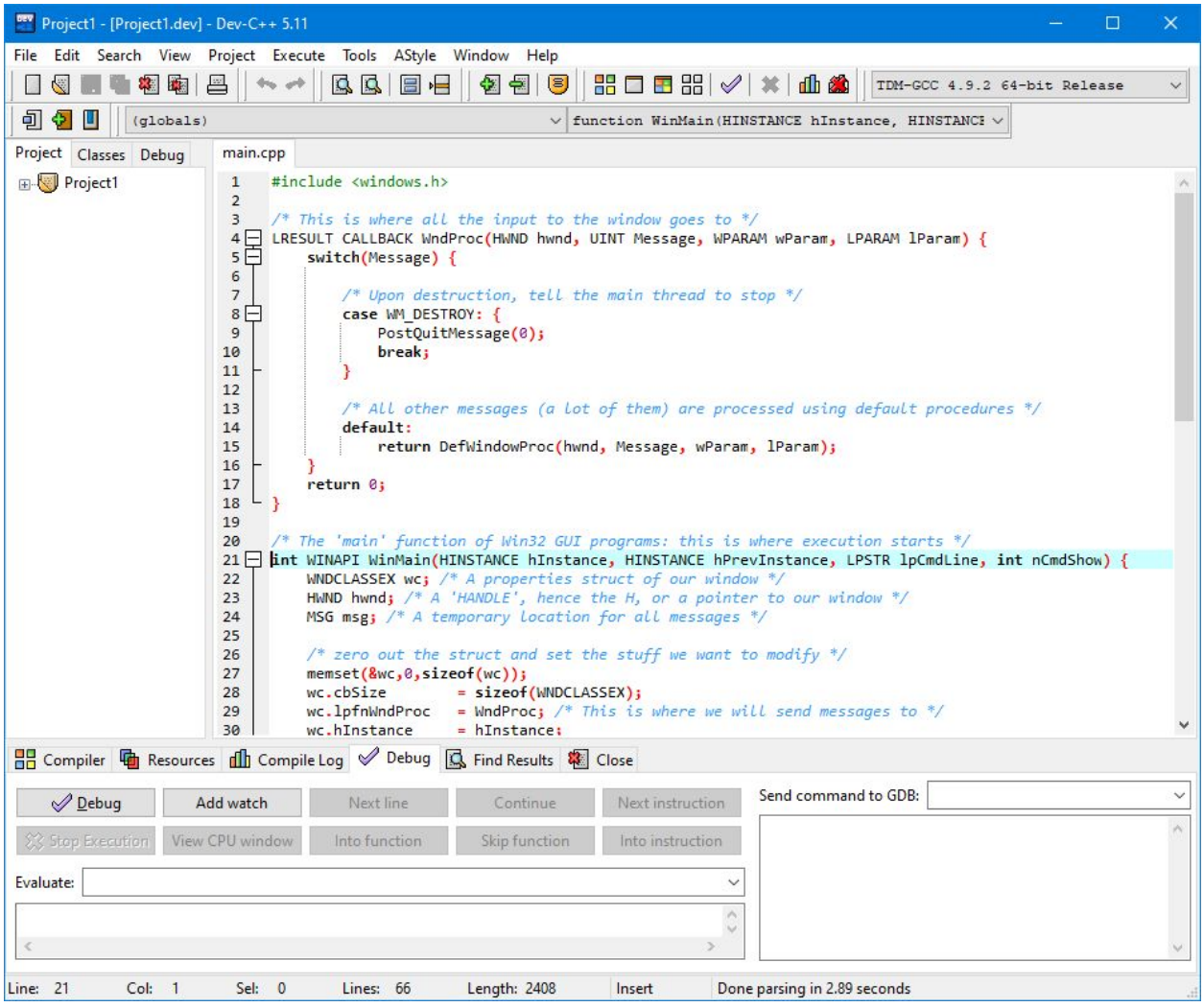
---

[1] Embarcadero. (2020). Embarcadero/Dev-Cpp. Retrieved December 03, 2020, from https://github.com/Embarcadero/Dev-Cpp

[2] Laplace, C. (n.d.). Home. Retrieved December 03, 2020, from https://www.bloodshed.net/

[3] Mes, J. (2020, November 30). Dev-C++. Retrieved December 03, 2020, from https://sourceforge.net/projects/orwelldevcpp/

A lot has changed in 20 years and Delphi 6/Delphi 7 both did not support Unicode and hence even the 5.11 version of Orwell Dev-C++ released in 2015 did not support Unicode. What this means is that under Dev-C++ 5.11 the default Delphi String type is an AnsiString. Starting in Delphi 2009 the default Delphi String became a UnicodeString. Migrating from AnsiString to UnicodeString can potentially be a significant undertaking but in this case, it was a pretty smooth upgrade process.



The Plan

There are a number of things that have to be thought of when upgrading an older project to the latest version. Some of these include:

- How much will it cost and is it worth the cost?
- Are the third-party components, tools, and libraries still around in an updated version or will new replacements have to be found?
- Is the codebase receptive to being upgraded or is it a mess of technical debt?
- Will the project benefit from the upgrade?

We'll answer these questions for the Dev-C++ project in this article, and provide you resources to answer those questions for your project as well.

Two Stages

It was determined that the modernization effort would take place in two stages. In the first stage, the minimum amount of changes would be made to get the project compiling in the latest version of Delphi. This was achieved with the Embarcadero Dev-C++ 5.50 release in a short amount of time. In the second stage, more modernization would take place to upgrade the compiler bundled with Dev-C++, upgrade its dependencies, upgrade its Unicode support, and add modern high DPI styling to fully support Windows 10. This was achieved with the Embarcadero Dev-C++ 6.0 release. As with any project, there is always more work to be done, and thanks to the update effort, that work will be much easier to complete.

Cost

How much will it cost and is it worth the cost?

In the case of the Dev-C++ project, the cost for upgrading it turned out to be rather low. The original budget ($9,000) allocated by the sponsorship was more than enough for the original upgrade from Orwell Dev-C++ v5.11 in Delphi 7 to Embarcadero Dev-C++ v5.50 in Delphi 10.4. The first stage used around $3,000 or 33% of the budget. Additionally, the original budget also satisfied the second stage with the upgrade to Embarcadero Dev-C++ v6.0. The second stage of the project upgrade used $6,000 or 66% of the overall upgrade budget.

Components and Libraries

Are the third-party components, tools, and libraries still around in an updated version or will new replacements have to be found?

The main third party components, tools, and libraries used and included in Dev-C++ are SynEdit, FastMM4, AStyle, and the compiler which was TDM-GCC 4.9.2. SynEdit is the core syntax highlighting editor control. FastMM4 is a custom memory manager that can be dropped into a Delphi project. AStyle is a C++ code syntax formatting utility. And TDM-GCC 4.9.2 is a custom bundle of libraries for Windows development paired with the GCC compiler. A number of other compressional libraries were also used by some of the Dev-C++ DevPack tools.

A newer version of SynEdit[4] was available (also on GetIt[5]) which included a number of breaking code changes. Some functions had been depreciated or moved plus changes in the Delphi compiler hid some of the Private properties. A newer implementation of FastMM4 was available as FastMM5[6] but the new version dropped in pretty much seamlessly. A new version of AStyle[7] was available with quite a few enhancements but it also dropped in seamlessly.

And finally, TDM-GCC[8] 9.2.0 was released early in 2020 with a newer GCC compiler featuring C++17 and partial C++20 support. Upgrading the project to TDM-GCC 9.2.0 went pretty smoothly with the main changes having to do with adding support for new command-line options. The

[4] Kassebaum, R., Vlahos, K., TurboPack, & et al. (2020). TurboPack/SynEdit. Retrieved December 03, 2020, from https://github.com/TurboPack/SynEdit

[5] Embarcadero. (2020). SynEdit for VCL. Retrieved December 03, 2020, from https://getitnow.embarcadero.com/SynEdit-1.5-Sydney/

[6] Le Riche, P., & Glienke, S. (2020, July 27). FastMM5 Version 5.02. Retrieved December 03, 2020, from https://github.com/pleriche/FastMM5/releases/tag/version_502

[7] Pattee, J. (2019, August 25). Artistic Style. Retrieved December 03, 2020, from https://sourceforge.net/projects/astyle/

[8] Eubank, J. (2020, February 16). Release v9.2.0-tdm64-1 · jmeubank/tdm-gcc-src. Retrieved December 03, 2020, from https://github.com/jmeubank/tdm-gcc-src/releases/tag/v9.2.0-tdm64-1

compression libraries used by the DevPack tools were largely replaced by TurboPack Abbrevia[9] (which is also available via GetIt).

Technical Debt

Is the codebase receptive to being upgraded or is it a mess of technical debt?

It can be tough to get a gauge of a project's true technical debt without diving into the project's code a bit more deeply. However, there are some tools that can help with this. One of these is built into the Delphi IDE and is called Method Toxicity Metrics[10]. It measures the language of functions, how many parameters they have, their If-Then statement depths, cyclomatic complexity, and finally gives a toxicity score for each function. Generally, the Dev-C++ codebase was pretty receptive to upgrading.

[9] Kassebaum, R., TurboPack, & et. al. (2020, November 25). TurboPack/Abbrevia. Retrieved December 03, 2020, from https://github.com/TurboPack/Abbrevia

[10] Embarcadero. (2016, March 23). Method Toxicity Metrics. Retrieved December 03, 2020, from http://docwiki.embarcadero.com/RADStudio/en/Method_Toxicity_Metrics

Method Name	Length	Parameters	If Depth	Cyclomatic Complexity	Toxicity
TDebugReader.GetAnnotation	91	1	40	45	5.263
FastMM_WalkBlocks	170	5	8	34	4.192
TFindForm.btnExecuteClick	123	1	9	44	4.071
TProject.BuildPrivateResource	168	1	3	29	3.808
TMainForm.LoadText	208	0	1	2	3.350
TImportMSVCForm.ReadCompile...	92	2	18	21	3.175
TExceptionFrm.ReadMapFile	91	1	3	27	2.763
TNewVarForm.btnCreateClick	93	1	2	24	2.663
TNewClassForm.btnCreateClick	99	1	2	18	2.487
TCompiler.WriteMakeObjFilesRules	71	1	4	21	2.263
TNewFunctionForm.btnCreateClick	69	1	2	21	2.237
TProject.AssignTemplate	78	2	3	16	2.142
TCompiler.Run	60	0	6	20	2.133
LogEvent	60	2	1	19	2.042
TTestClass.TestEditorList	97	0	0	1	1.962
TCompiler.WriteMakeDefines	69	1	3	14	1.946
TCompiler.Compile	73	0	1	12	1.913
TProfileAnalysisForm.DoGraph	65	0	2	14	1.896
TCompiler.GetCompileParams	50	0	4	18	1.875
TdevData.SettoDefaults	90	0	1	2	1.875
TdevCompilerSet.SetOptions	90	0	0	1	1.875
TDebugReader.HandleFrames	59	0	3	14	1.821
RunAndGetOutput	61	5	1	13	1.804
UpdateExpectedLeakList	54	3	4	15	1.800
TDebugReader.SyncFinishedParsing	56	0	3	14	1.783
TMainForm.CompEndProc	48	0	4	16	1.767
TProjectOptionsFrm.ButtonClick	32	1	2	20	1.733
TProject.LoadOptions	76	0	3	6	1.700
TFilePropertiesForm.ShowPropsFor	55	1	3	12	1.688
TProjectOptionsFrm.LoadText	74	0	0	1	1.675
TProject.Open	47	0	3	14	1.671
FastMM_ReallocMem_ReallocMe...	48	2	6	12	1.650
TProject.ExportToHTML	65	0	4	8	1.646
TProjectOptionsFrm.SetInterface	58	1	1	10	1.642
CreateOptions	38	0	3	16	1.642
TViewToDoForm.MatchesMask	41	2	1	15	1.637
TImportMSVCForm.ReadLinkerOp...	43	2	7	12	1.637

Benefit

Will the project actually benefit from being upgraded?

The answer to this question in the case of the Dev-C++ project is large, yes. Adding support for VCL styles and high-DPI is a huge step forward for the project. Modern code editors and IDEs

support dark mode and VCL styles allow the dark mode to easily be added to a VCL application like Dev-C++. There were complaints about Dev-C++ being blurry on today's modern Windows 10 systems and the high-DPI support in the latest versions of Delphi solves this issue. Deeper integration with source control and various CLI tools will also be a nice benefit for Dev-C++. Additionally, as the C++ language advances Dev-C++ needs to be able to keep up with language changes in C++20 and beyond.

The Team

A number of Delphi developers from around the world were brought in to implement the two stages of the plan. The upgrade was coordinated by an Embarcadero MVP with over 20 years of experience. The other developers were located using the UpWork freelancer site which makes it easy to hire Delphi (and other) developers for short term and long term projects. Developers from the Ukraine, Mexico, New Zealand, and the United States participated. One of the developers working on the project was even a user of the original Dev-C++ version back in college.

In addition to developers, a graphic designer was needed to head up the interface upgrade and a QA person was needed to test and verify all of the new functionality against the old functionality. Having both a graphics designer and a QA person really helped to increase the quality of both stages of the upgrade process. The QA person was able to test much more than the individual developers might have and caught bugs and regressions that otherwise would have slipped into the releases.

Upgrading The Code

The first thing that had to be done to upgrade this project from Delphi 7 to the latest version of Delphi was to determine how easy it would be to upgrade the project to compile in the new version. It took maybe an hour of search and replace to clean up all of the AnsiString and PAnsiChar references in the project with just String and PChar. There are some issues with this method where in some cases the AnsiString and PAnsiChar references had to be re-added due

to integration with the Win32 API causing mojibake[11] (the garbled text that is the result of text being decoded using an unintended character encoding, e.g. "🙀")

The First Compile

The next step to get the project to compile in the latest version of Delphi was to comment out any code referencing dependencies that had changed in either the Delphi RTL or third party projects such as SynEdit. The open-source SynEdit is a syntax highlighting library and a core component of Dev-C++. Each reference was commented out until the Delphi compiler allowed the project to compile. This process took around an hour.

The process was restarted around 6 different times. Each time more was learned about what various types needed to be updated and which did not. Additionally, each time more was learned about how to upgrade the code. On the final round of search and replace the TODO feature in Delphi was used to mark places in the code that needed to be upgraded or had been upgraded. This allowed using the TODO list IDE feature[12] to keep track of changed code so it could be verified by multiple developers.

TODO List

Several developers were then able to go back over the TODO list and customize and fix each piece of code as needed. In some cases, items marked by TODO were modified and then had to be double-checked by a second developer to make sure that the new code was correct. In other cases, the properties and functions of TSynEdit were private or protected. A helper class had to be used to access these properties and functions now under the latest version of Delphi. Accessing private properties and functions was a bug that was closed[13] in Delphi 10.1 Berlin.

[11] Mojibake. (2020, November 05). Retrieved December 03, 2020, from https://en.wikipedia.org/wiki/Mojibake

[12] Embarcadero. (2015). Using To-Do Lists. Retrieved December 03, 2020, from http://docwiki.embarcadero.com/RADStudio/en/Using_To-Do_Lists

[13] Cantu, M. (2016, June 8). Closing the Class Helpers Private Access Loophole. Retrieved December 03, 2020, from https://blogs.embarcadero.com/closing-the-class-helpers-private-access-loophole/

Pitfalls

Generally, everything went pretty well. However, as with any upgrade project, there were some unforeseen issues that had to be ironed out. One of these was that the TSynEdit third-party component used by Dev-C++ 5.11 had been customized (a fork basically of the original). This meant there were changes in that version that we didn't get by upgrading to the latest open source Turbopower SynEdit version. One of these was keyword highlighting of standard GCC terms. The Turbopower SynEdit version has keyword-highlighting setup for the Borland C++ compiler. This needed to be upgraded to support standard GCC keywords plus C++17 and some C++20 keywords.

The second unforeseen issue was with Unicode support and the UTF8 file format. By upgrading to the latest TSynEdit and using UTF8 by default some existing users had trouble with their ANSI based files because they were using versions of Windows that had a different code page set then the default English code page of Windows-1252. The end solution here was to bring full support for allowing users to select what encoding to open and save files as which is standard in most editors now. Delphi supports this through the TOpenTextFileDialog and TSaveTextFileDialog[14] dialogs. Marco Cantu has a Delphi and Unicode[15] document outlining some of the challenges with handling file encoding properly.

A final unforeseen issue was an attempt we made to bring in the TChromeTabs[16] third party component to replace the tab system for the editor. The TChromeTabs would have brought a close button on each tab and in general, are visually pleasing. It did not work out as planned as replacing the tabs proved to be too much of an effort for the allocated budget. We backed out this change and implemented the close tab buttons manually instead.

[14] Embarcadero. (2014). Vcl.ExtDlgs.TOpenTextFileDialog. Retrieved December 03, 2020, from http://docwiki.embarcadero.com/Libraries/en/Vcl.ExtDlgs.TOpenTextFileDialog

[15] Cantu, M. (2008, December). Delphi and Unicode. Retrieved December 03, 2020, from https://www.embarcadero.com/images/old/pdf/Delphi-Unicode181213.pdf

[16] Harazim, S., Thornton, P. S., & Et. al. (2020). Norgepaul/TChromeTabs. Retrieved December 03, 2020, from https://github.com/norgepaul/TChromeTabs

Upgrading The Interface

High DPI styles[17] are built into the latest version of Delphi which makes it easy to add both light and dark themes to an application. Additionally, the latest version of Delphi has support for Windows Per Monitor v2 built-in as well. A number of styles were chosen to bring a new look and feel to Dev-C++. However, in order to allow the user to select the styles additional work had to be done to bring this new functionality to the setup and configuration forms of Dev-C++. Lastly, the classic Windows style and icons were also kept in the project to allow for long time developers of Dev-C++ to feel right at home in the new version.

Upgraded High DPI Icons

The new light and dark themes also needed new icons that matched the new themes. This was achieved through the third-party SVGIconImageList component[18] which integrates with the Delphi image list features and makes it easy to add high-quality icons to an application. Each of the original icons in Dev-C++ was upgraded by a graphic designer to keep the original look of the icon and yet modernize it for the new themes. Only one version of the icons was needed because of a unique feature in SVGIconImageList which allows icons to be tinted to the color of your choosing at runtime. These new SVG tinted icons seamlessly integrate with VCL styles.

Upgraded Windows Dialogs

Dev-C++ is a file and code editor which means there are a number of standard Windows dialogs that it uses to open and save files. These dialogs are styled to the current theme by Delphi but in order to achieve the full effect of both light and dark themes a third-party library called VCL Styles Utils was used. VCL Styles Utils[19] extends full dark theme styling to these dialogs using the built-in high DPI styles.

---

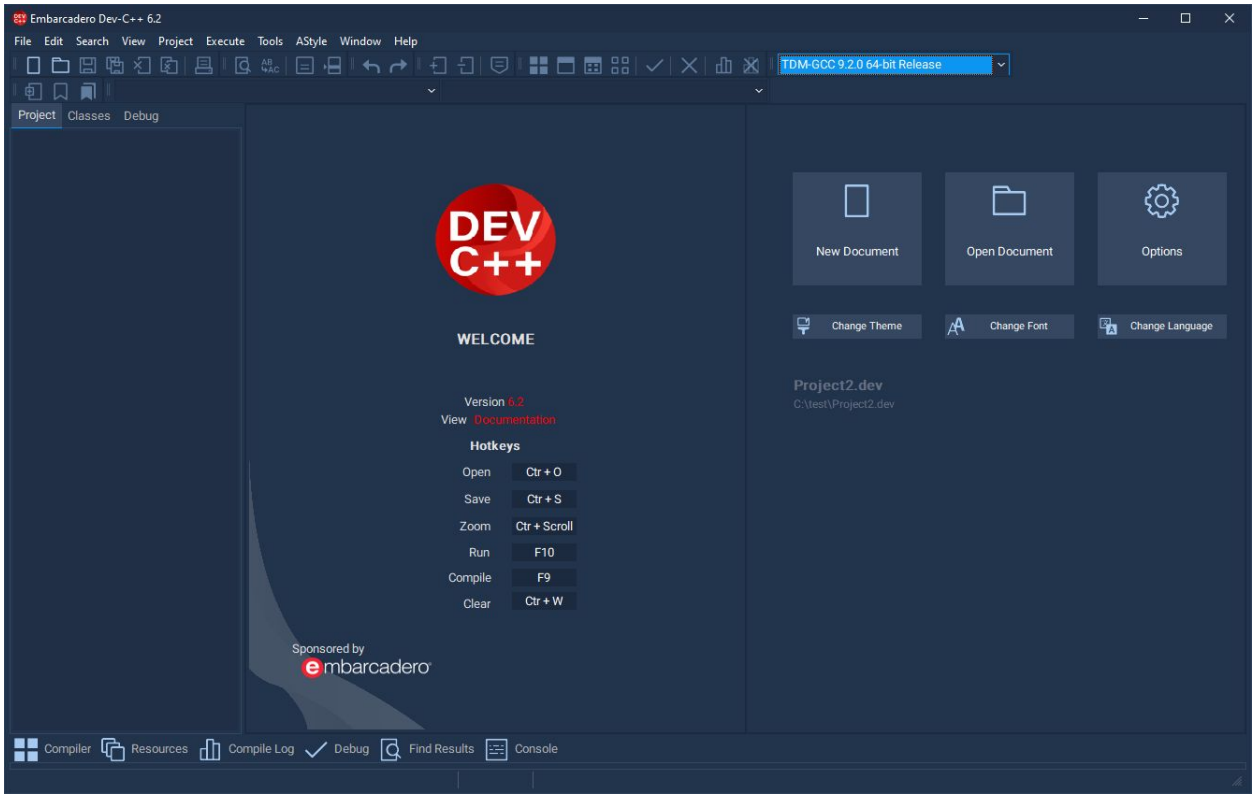
[17] Embarcadero. (2020, September 1). VCL Styles Support for High-DPI Graphics. Retrieved December 03, 2020, from http://docwiki.embarcadero.com/RADStudio/en/VCL_Styles_Support_for_High-DPI_Graphics

[18] Barazzetta, C., & EtheaDev. (2020). EtheaDev's SVGIconImageList. Retrieved December 03, 2020, from https://github.com/EtheaDev/SVGIconImageList

[19] Ruz, R. (2020, August). RRUZ/vcl-styles-utils. Retrieved December 03, 2020, from https://github.com/RRUZ/vcl-styles-utils

New Welcome Screen

Modern code editors and IDEs have a Welcome screen that appears when the user first launches the application or when no code windows are open. Dev-C++ was lacking a screen like this and so one was implemented. This new Welcome screen includes quick help like shortcut keys, links to documentation, and some new fast navigation buttons for opening new and recent files.



A Few New Features

In addition to the upgrades, we also wanted to throw in some new features that help to define high productivity in a modern IDE. The main feature we added was to have Windows CMD and PowerShell embedded shells within the IDE itself. These are tabbed allowing multiple CMD and PowerShell instances to be opened within the IDE similar to how the new Windows Terminal works. A second feature we added was placing a close button on the right side of each editor tab. This allows a user to quickly close a tab with only one click (verses 2 clicks via the popup menu).

A few new Editor styles (coloring of the text in the code window) have been added which people enjoy as well. Plus per a user request font ligature support was added by a maintainer to TSynEdit and we were able to pull that change down into Embarcadero Dev-C++ as well. This is on top of the high-DPI goodness that comes built-in with the latest version of Delphi and the VCL styles that were added as mentioned earlier.

A Beautiful Modern Open Source C++ IDE

Dev-C++ is a popular C++ development and learning environment on Windows but its legacy codebase has been holding it back a bit with no recent updates to support all of the latest features of Windows 10. The modernization effort was achieved within the original sponsorship budget allocated to the project and the new high DPI VCL style enabled interface looks great.

The new modernized codebase of the main Embarcadero Dev-C++ project is ~180,000 lines of code which compiles in 6.5 seconds under the latest version of Delphi. The new TDM-GCC 9.2.0 compiler brings C++17 and partial C++20 support to the project. Modernizing the codebase and interface has really breathed new life into the future of the project and set it up to take advantage of modern advances in Delphi, Windows, high-DPI, C++20, and beyond.

If you love C++ and Delphi for Windows development head over and take part in the project. We can't wait to see what is in store for Dev-C++ next!

- Download
  - www.embarcadero.com/free-tools/dev-cpp/free-download
- GitHub
  - github.com/Embarcadero/Dev-Cpp
- Blog Posts
  - blogs.embarcadero.com/tag/dev-cpp/
- Migration & Upgrade Center
  - www.embarcadero.com/rad-in-action/migration-upgrade-center