

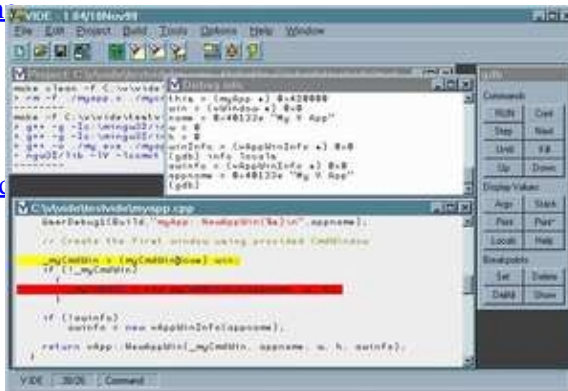
VIDE

Table of Contents

<u>VIDE User Guide</u>	1
<u>VIDE Quick Start Guide</u>	11
<u>VIDE for Windows with gcc (MinGW/Cygnus)</u>	12
<u>VIDE for Windows with Borland BCC 5.5</u>	16
<u>VIDE for Linux with gcc</u>	20
<u>VIDE – Command Reference</u>	24
<u>VIDE – Editor Reference</u>	39
<u>VIDE C/C++ Tutorial</u>	58
<u>Using VIDE with the Sun JDK</u>	68
<u>The Borland C++ Compiler 5.5</u>	80

V IDE User Guide

- [The V Integrated Development Environment](#)
- [VIDE Overview](#)
- [VIDE Help System](#)
- [Debugging with VIDE for MS-Windows](#)
- [VIDE for Linux](#)
- [Release Notes](#)
- [Installing VIDE](#)
- [Known VIDE problems](#)
- [Help make VIDE better!](#)
- [No Warranty](#)



The V Integrated Development Environment

VIDE is the V Integrated Development Environment for GNU gcc (Gnu compiler collection), the free Borland C++ Compiler 5.5 for MS-Windows, and the standard Sun Java Development Kit. VIDE is available both as a ready to run package for MS-Windows 9x/NT and Linux, and as part of the **V C++ GUI Framework**. Executables for MS-Windows 9x/NT and Linux (glibc) are available for download at <http://www.objectcentral.com/vide.htm>.

If you want to get started using VIDE with C or C++ as quickly as possible, please see the [VIDE Quick Start Guide](#) for instructions on installing VIDE and creating your first VIDE project.

VIDE has been designed by a programmer for programmers. It makes the task of developing software for C/C++, Java, and HTML much easier than using command line mode. It is easy to learn, so it is a good tool

VIDE

for the beginner. It also has the critical features needed to enhance the productivity of the experienced programmer.

The source code is available under the GNU Public License (GPL), and many parts of its design reflect the philosophy of GNU and Open Source. Whenever possible, VIDE takes advantage of existing GPL or freely available software. It is designed to use the GNU gcc compiler and the free Sun Java kit. It also uses the GPL ctags program, and the addition of more integrated support for other GNU tools is planned.

VIDE has intentionally followed a minimalist design philosophy. The traditional development environment long favored by many programmers has been to work with each tool individually from a command shell. Thus, the programming environment would include an editor (usually vi or emacs), a compiler, a linker, a make tool, and a debugger. On Unix systems, all the other standard software tools would also be available – grep, ctags, lex, yacc, and so on. With this approach, the developer controls everything through command windows.

VIDE tries to follow this tradition in many respects, while allowing the programmer many of the conveniences of an IDE, and of using a true windowed interface. What this philosophy means in practical terms is that you, the programmer, need to be more aware of what is going on with the tools VIDE supports. VIDE will automatically provide the most common options, but to get at the full power of the underlying compiler, you should understand how a command line compiler works – which means understanding how it uses command line switches. You may also need to understand a bit of how the make tool really works to get the full power of building projects with makefiles.

It is entirely possible to work with VIDE and never get down to the nitty gritty of the compiler and make program. It is easy to create a new project, add source files, use the standard compiler options, and compile and debug your project without ever needing to alter VIDE defaults. However, you may find that what is going on underneath is not as hidden as it might be in other IDEs, especially when there is a problem such as a missing file or library. You will need to understand what the error messages from the compiler or linker mean.

While VIDE doesn't have every feature found in many commercial development systems, it is an ongoing project, with more features included in each release. And best of all, VIDE is *free*! And since it is GPLed, you can help add even more features if you want.

The main features in the current release of VIDE include:

- **A great editor** – The VIDE editor is a very good editor designed for the programmer. Editor features include:
 - ◆ Syntax Highlighting for C/C++, Java, Perl, Fortran, TeX and HTML.
 - ◆ Several menu-selectable editor command sets, including:
 - ◇ A **generic** modeless command set, similar to many Windows editors.
 - ◇ **Vi** – the standard Unix editor, with extensions.
 - ◇ The **See** editor command set, an editor designed and used by Bruce Wampler, the author of **V** and VIDE.
 - ◇ Others easily added by extending a C++ class.
 - ◆ Beautifies C/C++ and Java code
 - ◆ Powerful command macro capability
 - ◆ Complete support for ctags (symbol lookup)

The Editor Reference has a short section of [tips](#) that will help you get the most out of the editor.

VIDE

- **Project Files** – specify source files, compiler options, and other details required for g++ or Java. Project files simplify and hide most of the details of using the underlying tools.
- **gcc and Sun JDK** – Supports development of both C/C++ with the GNU **gcc/g++** compiler for MS–Windows and Linux, (OS/2 environment soon), as well as the Java development using the **Sun JDK**.
- **Borland C++ Compiler 5.5** – VIDE has very good support for the recently released free Borland command line compiler tools. It can build Console, GUI, and static library project files. I also includes some extra documentation about the Borland environment. VIDE works with the recently released Borland Turbo Debugger. Note that TD32 still has some bugs, and does not seem to properly debug apps developed using the V GUI.
- **Building Projects** – Uses standard GNU *make* to build projects for g++, and the standard features of the JDK to build Java projects.
- **Syntax errors** – Point and click to go to errors in source files.
- **Supports gdb and jdb** – Integrated support for the GNU *gdb* debugger for C/C++ and Sun's *jdb* debugger for Java. The most common debugging tasks, such as stepping through a program, are fully integrated, yet all the more advanced features of *gdb* and *jdb* are available through a command line window.
- **V GUI** – Integrated support for the V GUI for C++, including the V app generator and the V icon editor.
- **HTML** – Extra support for HTML development. While VIDE doesn't support WYSIWYG HTML development, you can send the current HTML file to your browser for immediate viewing. A comprehensive HTML help document is included. Future versions will include more HTML features such as table generation and image sizing.
- **The VIDE Help System** – Includes extensive HTML based help. Covers VIDE, GNU utilities, C/C++ libraries, HTML, and more. The help files are available for separate download. You can see a complete online version of the help package [here](#).
- **Future enhancements** – VIDE is under active development and will continue to improve. Additions planned include an interface to CVS version control support, spelling checking, C++ class browser, gcc profiler interface, and other features as supplied or requested from the VIDE user community.

The executable version of VIDE is totally freeware. Use it, share it, do whatever you want. The source of VIDE falls under the GNU General Public License, and is normally included with the V GUI distribution. See the file COPYING included with the distribution for more information. Its development is not always in phase with the current V distribution, so there will be additional releases of executable versions as they become available. With the added support for Java, it is likely that the standalone executable version will see broader use than the source version included with V.

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

Features that are planned for the near future include:

VIDE

- Support for more editor command sets, including Emacs.
- Integrated spelling checking.
- Support for CVS/RCS.
- Code templates.
- Predefined Project files for various configurations – GUI, console, Mingw32, Cygnus, etc.

This document describes how to use VIDE. Because the process is slightly different for C/C++ and Java, there is a brief tutorial section for each language. Following the tutorials, all the commands available from the menus are described.

VIDE Overview

The design of VIDE has been somewhat evolutionary, but you should find that it is not that much different than other IDEs you may have used. Because VIDE is a free, open source program, it probably lacks some of the polish of commercial IDEs. However, it is still quite functional, and it is really easier to develop programs with it than it is to use a command line interface.

Generally, any application you write will consist of various source files (with associated header files for C/C++), and required data files. These files are generally dependent on each other. By defining a VIDE Project for your application, all the file dependencies are automatically handled. The standard tool *make* is used for C/C++ files, while the JDK Java compiler automatically handles dependencies.

Using VIDE, the normal work cycle goes:

1. Design your application.
VIDE currently has no capabilities to help with this stage.
2. Start VIDE, and create a Project File.
This will include all source files, compiler options, and other information needed to compile your application.
3. Build your project.
This stage compiles your source into object code. Compilation errors are displayed in the status window, and you can simply right-click on the error to go to the offending line in your source code. After making corrections, you repeat this step until all compilation and linking errors are removed.
4. Run your program.
You can start your program from within VIDE.
5. Debug your program.
VIDE for MS-Windows has integrated support for the **gdb** debugger for GNU C/C++. Because the *DDD* debugger available for Linux is so good, VIDE for Linux does not have integrated debugging support, but will automatically launch DDD. There is no support for debugging Borland BCC32 programs.
6. Write documentation for your application.
VIDE has syntax highlighting for HTML to make that job easier. You can also automatically launch your web browser to view the resulting HTML pages. Really neat.

VIDE Help System

VIDE is distributed with this complete VIDE documentation. A complete set of HTML documents with useful help topics are available at www.objectcentral.com. VIDE also knows about the documentation that

comes with the Sun Java distribution. All this help is easily available from the VIDE Help menu. The vide help system is described in the [VIDE command reference](#).

Debugging with VIDE

VIDE supports GNU **gdb** and the Sun JDK **jdb** debuggers. The VIDE interface to the debuggers has been designed to make the most common debugging tasks easy. The goal is to make using the native debuggers as easy as possible for casual users, while maintaining the full power of the debugger for experienced users. VIDE accomplishes this by showing a command window interface to the debugger. You can enter any native debugger command in this window, and thus have full access to all debugger features.

VIDE makes using the debugger easier by providing an easy to use dialog with the most often used commands. Breakpoints are highlighted in yellow. And as you debug, VIDE will open the source file in an editor window and highlight in red the current execution line on breakpoints or steps. It is very easy to trace program execution by setting breakpoints, and clicking on the *Step* or *Next* dialog buttons. VIDE also allows you to inspect variable values by highlighting the variable in the source and clicking the print button.

VIDE for MS-Windows

VIDE for Windows is distributed as a self-installing executable file. You might want to create a desktop icon to start VIDE. As of 1.08, VIDE supports drag and drop to edit files. You can make VIDE the default editor for your source files by using the Windows command: **Start**→**Settings**→**Folder Options**→**File Types** dialog to associate your source file types (e.g., .c, .h, .cpp) with VIDE.

VIDE for Windows supports two compilers: the GNU gcc compiler in both the MinGW and Cygnus versions, and the free Borland BCC 5.5 compiler. The gcc C/C++/Fortran compiler is quite mature, and works quite well with VIDE. Borland's free compiler is a relatively new release, and all the quirks have not yet been fully documented. Please check the [VIDE Borland](#) reference guide for details on using VIDE with BCC 5.5.

You can use a `-p=PrefFile` switch on the startup to specify an alternate `PrefFile.ini` file for VIDE preferences. This makes it easier to use both the GNU gcc compiler and the Borland BCC 5.5 compiler on the same system. Add this switch to the "Target:" field of the Shortcut tab of the Properties menu you get when you right-click the desktop icon.

VIDE for Linux

The Linux version of VIDE is distributed as a semi-statically linked binary version for recent versions of Linux. It is based on the new Open Motif version of V, and is statically linked to the V and Open Motif libraries, but dynamically linked to the C/C++ libraries. To install, unzip and untar the distribution. You can install the binary almost anywhere you want.

Because Open Motif doesn't know how to interact with Gnome properly, V apps will start with the default Motif decoration colors. You can get a different color schemes by using the `-bg` startup switch. This changes all the Motif decorations to be based on the color you specify. For example, starting VIDE with `VIDE -bg gray75` gives a nice gray based color scheme. You can make a desktop shortcut with either KDE or Gnome that will automatically use the `-bg` switch.

Release Notes

- **Version 1.21 – 28Feb2001**

This version has a new feature – the %N operator for the Tools menu (this was useful for TeX users, and there was a 1.20 alpha release of VIDE that was never "official".), and a small bug fix when Java files had a leading "/" in syntax error messages.

- **Version 1.19 – 10Oct2000**

This version has a small bug fix – starting VIDE by double clicking a Windows .vpj file did not start in the correct directory. It also adds tool bar buttons to start the first and second defined tools on the Tools menu.

- **Version 1.18 – 22Sep2000**

This version makes the help files defined as an option rather than hardwired. An option for defining how to run tools was added.

- **Version 1.17 – 15Sep2000**

This version adds some enhancements. The most significant is that it allows the definition of up to eight external tools that can be run from the Tools menu with various options and file names. This version also supports multiple end of line formats. The documentation has some significant revisions, including the new [VIDE Quick Start Guide](#).

- **Version 1.16 – 15Jul2000**

This version was never really properly released. It was available as source code, but never really supported.

- **Version 1.15 – 15Jun2000**

This version adds support for Borland's Turbo Debugger and gdb 5.0. There were a few minor bug fixes as well.

- **Versions 1.12 to 1.14 – 01Jun2000**

These versions have been bug fixes and fine tuning, some just for Borland support.

- **Version 1.11 – 18Apr2000**

What happened to the release notes for 1.09 and 1.10? They never were entered! Version 1.10 was the most stable X version to date, and it may be some time before a 1.11 X binary X version is released. Version 1.11 has some very significant additions that are most useful for the MS–Windows platform.

- ◆ *Print support* – A print command has been added to the File menu. It will make a reasonable printed copy of your source file. No syntax highlighting yet.
- ◆ *New Build Process!* – The way VIDE runs make and builds projects on MS–Windows has been completely redone. VIDE now uses pipes rather than intermediate files to get the output of the make. This really makes VIDE work better with Borland's BCC 5.5. The output is now shown as it happens, rather than when the whole make is done. The other major difference is that you can now continue to edit while the make is going on.
- ◆ *New start up switch* – You can use a `-p=PrefFile` switch on the startup to specify an alternate `PrefFile.ini` file for VIDE preferences. This makes it easier to use both the GNU gcc compiler and the Borland BCC 5.5 compiler on the same system.

- **Version 1.08 – 04Mar2000**

Version 1.08 is an important release. It corrects some bugs in the Borland BCC32 5.5 interface. It adds drag and drop for the MS–Windows version. But it also finally supports the Motif Linux version. This is the first binary release in some time for a Linux version. It is also important because it corresponds to the finally released V GUI Version 1.24, which contains the source for VIDE 1.08.

- **Version 1.07 – 25Feb2000**

Version 1.07 is a major upgrade. The main new addition is support for the new free version of the Borland C++ Compiler 5.5. In the process of adding this support, some features were improved.

- ◆ *BCC32 support* – fully integrated to VIDE. A separate [reference document](#) for the Borland support has been added.

VIDE

- ◆ *Easier resource files* – It is now easier to include `.rc` resource files on Windows.
- ◆ *Improved Project Editor* – The project editor is a bit more intelligent about rebuilding the Makefile now, and won't automatically regenerate it as often.

• Version 1.06 – 8Feb2000

Version 1.06 has several significant new features added. Gee, this thing is starting to get real!

- ◆ *ctags* support – you can easily find the declaration or definition of any program symbol.
- ◆ *C++ Project Wizard* – When you create a new C++ project, you can easily specify options such as project type (console, GUI, library), compiler options, and more.
- ◆ Significant upgrades of the documentation, including a new editor tips section.
- ◆ Syntax highlighting added for Fortran. There is no project support for Fortran, however.
- ◆ Syntax highlighting for LaTeX files. Additional support for LaTeX is likely in the future.

• Version 1.05 – 21Jan2000

- ◆ Autoindent for code files has been added. This option is found on the Options->Editor dialog.
- ◆ Code beautifier now supports KRbrace placement in addition to braces on separate lines.
- ◆ Syntax highlighting added for Perl. There is no project support for Perl, however.
- ◆ Screen updating for Windows version significantly faster.
- ◆ A few bugs have been fixed.

• Version 1.04 – 18Nov1999

Version 1.04 includes major enhancements to the debugger interfaces. Debug commands have been removed from the tool bar, and now are in a pop up dialog when you run the debugger. The biggest improvement is that breakpoints are now highlighted in yellow in the source file windows. The current execution line is shown in red. The Debug menu has been removed.

With the release of gdb 4.18, programs developed with the V GUI can now be debugged reliably. With the release of gcc 2.95 for mingw32, and the new Cygnus Version 1.0, all the problems associated with earlier releases seem to have been resolved. You should be using the latest versions of gcc!

• Version 1.03 – 25Oct1999

This version has some minor enhancements with the gdb and jdb interfaces. The documentation HTML has been changed to use the default browser background colors.

• Version 1.02 – 11Oct1999

This version includes new support for different syntax highlighting color schemes and different background colors. There are six choices under the Options:Editor menu dialog.

Installing VIDE

Executable versions of VIDE are available for both MS-Windows and Linux. The MS-Windows is a self-installing executable, while the Linux version is a gzipped tar file.

VIDE for Windows

After you've downloaded the VIDE setup package, simply run it. You will be given the choice of where to install VIDE. The default is usually fine, but you may want to install it to the same binary directory that your compiler is on.

After you've run setup, you still have two steps.

Create a Desktop Icon

You can run VIDE from the Start menu if you wish, but it is often more convenient to add a desktop icon. Use Explorer to find the VIDE executable, then right-click to create a desktop startup icon for it.

Setting Environment Path

For VIDE to work properly, you *must* have your Windows PATH environment variable set correctly. The PATH must contain both the path to VIDE, and the path to your compiler. To add the VIDE directory and the compiler's binary directory to your PATH, on Windows9x, edit the file `C:\autoexec.bat`. Simply add the compiler's binary directory to the PATH command. On NT, you use the system settings menu off the Start menu to change the PATH in the environment.

VIDE with gcc – MinGW or Cygnus

By default, VIDE will work with the GNU gcc compiler. The only requirement is that the gcc binary path be included on the PATH as described above.

VIDE with Borland BCC 5.5

VIDE works quite well with BCC 5.5. However, you *must* have BCC 5.5 set up correctly first. You also must explicitly tell VIDE where the Borland compiler is. (This is because gcc is supported by default, so you must let VIDE know that you are using Borland's BCC compiler instead.) All the details for setting up BCC 5.5 and using it with VIDE are described in the [Borland BCC 5.5](#) guide. Note especially the following two sections:

- [Setting up BCC 5.5](#)
- [Using Borland C++ with VIDE](#)

VIDE for Linux

The Linux version of VIDE is distributed as a statically linked binary version for recent versions of Linux. It is based on the Motif version of V, and is statically linked to the MetroLink Motif library. To install, unzip and untar the distribution. You can install the binary almost anywhere you want.

General Installation Notes

If you want to use the help files other than VIDE help, you will need to download the separate help file archive. It is best to install the help files in the `/vide/help` directory. After you've installed the help files, you will need to use the VIDE Options:Editor command to set the path to the help files.

VIDE does not provide Java help. You will need to find and download the Sun JDK yourself. Then use the VIDE Options->Editor menu command to set the Java directory.

If you are building VIDE from the V distribution, then apply the above comments to the version you build. If you want to use the V appgen program, or the V Icon Editor, you need to get those from the V distribution.

Known VIDE problems

For the X version, errors generated by the compiler are passed via two temporary files. Sometimes these files are not deleted, and are left behind on the hard disk. If two instances of VIDE are running, it is possible to have an access conflict to these files from one of the instances. This can prevent builds with error messages from displaying those messages in the message window.

If g++ has a fatal error, it can hang VIDE. Because all files are saved right before running g++, there is no data loss.

The syntax highlighting doesn't work right for multi-line `/* comment */` style comments. If the lines between the opening `/*` and the closing `*/` have a `"` as the first character (which is the normal convention for Java programs!), syntax highlighting works correctly. Otherwise, those lines will not get the comment highlight. This is purely a cosmetic problem, but is unlikely to be fixed because of difficulties imposed by the internal structure used by the editor. (Note (3Sep99): Gee, I feel better! I was just playing with Microsoft VC++ 6.0, and guess what? Their editor doesn't do `/* */` comments right, either! I guess if Microsoft can sell a commercial product that can't do proper syntax highlighting, then I can do it for a free product.)

The tidy/prettypoint command doesn't handle the last case of a switch properly. This won't be fixed. Also note that the formatting is based on the indent of the previous line, so you must start at a known good indent point.

Help make VIDE better!

The V IDE is GPL freeware. It has been written using the V C++ GUI framework. I get no compensation for either V or VIDE, so please don't get too fussy about features or problems. I welcome bug reports and requests for new features, but I make no promises.

My goal for VIDE is to make it a great alternative to Emacs. I know Emacs will do almost anything you would ever want, but the learning curve is huge. VIDE is much more GUI oriented, and I want to keep it simple enough for beginning programmers to use.

So, even more welcome than bug reports would be offers to help add features to VIDE! Since the source code for both V and VIDE are GPL and LGPL, they are available for enhancement. If you would like to make contributions to VIDE, please contact me directly by e-mail. VIDE is currently ahead of the V GUI release cycle, so I will need to provide you with the latest versions of all the source code.

I welcome any contributions or ideas, but right now the following projects:

- Support for other editor command sets. My goal is not to provide exact clones of other editors, but to provide the basic commands of other editors at a "finger memory" level. It is fairly easy to add a new editor command interpreter. I spent less than two days writing the Vi emulation. Because I use the See command set myself, it is a bit hard for me to write a command interpreter for other editors that provide true "finger memory" compatibility. I hope the Vi emulation meets this kind of compatibility. I'd like to see an emulation for Emacs, but any emulation is welcome. I think the emulation will be better if it is written by someone who actually uses that editor.
- Spelling checking.
- An interface to CVS.

VIDE

- A class browser for both C++ and Java. I have a stand alone V C++ browser that could serve as a starting point.
- An interface to other Java tools such as Javadoc.
- An interface to Jikes.

If you'd like to help on any of these projects, please contact me and I will do everything I can to help you get started.

No Warranty

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

V IDE Reference Manual – Version 1.11 – 19April2000

Copyright © 1999–2000, Bruce E. Wampler

All rights reserved.

Bruce E. Wampler, Ph.D.

bruce@objectcentral.com

www.objectcentral.com

VIDE Quick Start Guide

VIDE 1.23 – 24Jul01

[VIDE User Guide](#)

This document is intended to get you started using VIDE as quickly as possible with your compiler. However, we strongly recommend that you read the full [VIDE User Guide](#) to get a complete overview of VIDE's capabilities. *If you are seeing this as a result of running the VIDE installation program, you can now click in the install window, and finish the installation.* This will remain after the install is complete.

- VIDE for Windows with gcc (MinGW/Cygnus)
 - ◆ [Installing VIDE with gcc](#)
 - ◆ [Your first VIDE project](#)

 - VIDE for Windows with Borland BCC 5.5
 - ◆ [Installing VIDE with Borland](#)
 - ◆ [Your first VIDE project](#)

 - VIDE for Linux with gcc
 - ◆ [Installing VIDE](#)
 - ◆ [Your first VIDE project](#)

 - [VIDE for Sun Java \(JDK\)](#)

 - [No Warranty](#)
-

VIDE for Windows with gcc (MinGW/Cygnus)

Installing VIDE for Windows with gcc [top](#)

- **Install the compiler** Your first task is to be sure your compiler has been correctly installed. A good test is to run a Command Window (Usually **Start**→**Programs**→**MS-DOS Prompt**), and then enter the command "gcc -v" to display the version message from the compiler. This should work for both MinGW and Cygwin versions of gcc. If you don't see the gcc version message, then you most likely need to add the gcc directory to your environment path.
- **Install VIDE executable** After you download VIDE, install it onto your system. It is best to install VIDE to the default directory (/Program Files/vidé). The installation will create an entry on the Windows Start menu, but you may also want to create a desktop icon for VIDE.
- **Set environment PATH** In order to use the tools that come with VIDE (e.g., ctags), the directory you installed VIDE to needs to be on your environment PATH. (Your compiler must also be included in the PATH.) To do this on Windows 98, you must edit the file C:\AUTOEXEC.BAT. You should find a line in it that begins with "PATH=". If you've installed VIDE in the default location, you can add "C:\PROGRA~1\VIDE" to the path (note the "PROGRA~1" instead of "PROGRAM FILES"), separated from other path directories by a semicolon. On NT and 2000, you use the system settings menu off the Start menu to change the PATH in the environment.
- **Set VIDE options** VIDE has several options that you might want to set to make it work best for you. Most options are found from the "Options" menu after you start VIDE.
 - ◆ The default values from Options→VIDE dialog are generally fine for gcc.
 - ◆ You can use Options→ to set various attributes for the editor, including command set, indentation, and syntax coloring.
 - ◆ The Options→Font menu allows you to set the monospaced font of your choice. If you ever find that the cursor in VIDE does not work properly, it usually means you've re-installed VIDE or altered the font set on your machine. Simply reselect your font if this ever happens.
 - ◆ You can make VIDE the default editor for your source files by using the Windows command: **Start**→**Settings**→**Folder Options**→**File Types** dialog to associate your source file types (e.g., .c, .h, .cpp) with VIDE.

Your First Project for Windows with gcc [top](#)

VIDE can be used simply as an editor to edit any text file such as a C++ program or an HTML file. Simply use the File menu to open a file of your choice. For a programming project, however, VIDE lets you create a project that makes working with your program much easier. The following steps describe how to create a project, and then how to compile your program.

VIDE has been designed to use the standard tools that come with your compiler. Before compiling your program, VIDE will use the project options you've set to generate a Makefile to be used with the standard gnu

make program. VIDE then runs make to compile your project. For most cases, you won't need to know the details of how VIDE generates the makefile.

- **Create a New Project**

The first step is to create a new project by using the **Project→New C++ Project** menu command. You will be prompted for the name of your project. Generally, you will create your project file in the same directory that contains your source code.

Once you have selected the project name, you will see a dialog that lets you define important characteristics of your project. The options you set in this dialog determine some important default options used to generate the Makefile. First, select if you are generating a C or C++ program. You can mix C and C++, but you would have a C++ project. Fill in the Target name box. This is the name of the executable or library file you want to create, including an appropriate extension (such as .exe).

Next, select the type of application you are building. Your choices include a "Console Application", which is one that does not use the Windows GUI API, such as the standard "Hello, World" app. If you are going to build a Windows GUI application, select "GUI Application". This allows you to use the standard Win32 API. Note that gcc does not include any advanced GUI libraries such as MFC. If you are using the V GUI library or OpenGL, check the appropriate box.

Your other options in the New Project dialog box include if you want to generate a Release version or a Debug version. (VIDE does not include an automatic option for switching back and forth between Release and Debug. You can either create two projects, or manually change the compiler switch from -g to -O.) Finally, select the compiler. MinGW will be the default. You also have an option to select the Cygnus compiler, and to use the -no_cywin switch.

- **Use Project Editor to Add Files**

After you've made the appropriate selections in the New Project dialog, the new project will be opened in the Project Editor. This lets you edit all the properties of your project. You can come back to the Project Editor at any time from the **Project→Edit** menu.

Typically, the first thing you should do is add the source files included in your project. Select the Files tab of the Project Editor, and add all the source files used by your project. (You only add the C/C++ source files, and not the .h header files.) If your GUI project has any resource files (.rc), include them, too.

If you don't yet have a source file, you can skip this step for now, and come back later to add them by re-opening the Project Editor with the **Project→Edit** command.

- **Set Paths**

If your project uses any libraries or include files that aren't on the standard compiler search path, pick the Paths tab to add these paths. Any paths you include from this dialog will be added to the makefile. You can also keep your source, object, and executable files in different directories if you want, although that is usually not necessary. It is easiest to keep your project and program files in the same

directory, usually the current directory (.).

- **Set Defines**

If you need to pass any defines to the compiler, pick the Defines tab. First, add defines to the pool list (in the form `-DFOO` or `-UFIE`), then add those to the active list. Many programs will not need to use this feature.

- **Add Libraries**

If your program uses any non-standard libraries (most of the Windows API libraries are automatically included by gcc), select the Files tab and add them to the Linker Flags box. For example, you could use the common controls library by adding `"-lcomctl32"` to the Linker Flags line. If you get a lot of undefined symbols when you compile your program, it is likely that you need to include some library on this line. It is up to you to understand which libraries your program needs.

- **Set Compiler Options**

Many programs will compile and run correctly using the default compiler settings. However, the compiler has many options that you may want to or need to use. Add any switches the compiler needs to the Compiler line of the Files tab dialog.

- **Close the Project Editor**

Once you've added all the source files, and set up whatever other options you need to, close the Project Editor. You can edit the Project later from the **Project**→**Edit** tab. When you next run VIDE, you will need to re-open your project using the **Project**→**Open** command.

- **Edit your files**

After you have created your project, you can use the VIDE editor to edit any of them as needed.

- **Build your project**

Once you've defined your project and have all the source files edited and added to the project, you can compile your program. You can use the "build" button on the tool bar, or select an option from the **Build** menu. Remember that VIDE generates a standard gnu makefile, and runs `gnu make` to do the actual building of your project.

The results of the build are shown in the VIDE message window. If there are syntax errors in your source files, you can simply right click on the error message with the line number to open that file and go directly to that line. Fix your syntax error, and try to build again.

VIDE

At this point, you've now used the basic features of VIDE. You can read about other VIDE features, such as using the debugger, in the [VIDE User Guide](#).

VIDE for Windows with Borland BCC 5.5

Installing VIDE for Windows with Borland BCC [top](#)

VIDE is by default setup to work with the `gcc` compiler. *For it to work correctly with the Borland BCC compiler, it is **essential** that you have set up VIDE for BCC as described in the following steps.* If you don't follow these steps, you won't be able to successfully compile programs from within VIDE.

- **Install the compiler** Your first task is to be sure your compiler has been correctly installed. A good test is to run a Command Window (Usually **Start**→**Programs**→**MS-DOS Prompt**), and then enter the command "bcc32" to display the help message from the Borland compiler. If you don't see the BCC help message, then you most likely need to add the BCC bin directory to your environment path. It is also very important that you've properly created the `.cfg` files for the compiler and the linker. More details about installing BCC can be found in the full VIDE [Borland](#) help file. That file has some very useful information about BCC 5.5 that is not clearly explained in the standard BCC 5.5 documents, so you should be sure to read it carefully.
- **Install VIDE executable** After you download VIDE, install it onto your system. It is best to install VIDE to the default directory (`/Program Files/vid`). The installation will create an entry on the Windows Start menu, but you may also want to create a desktop icon for VIDE.
- **Set environment PATH** In order to use the tools that come with VIDE (e.g., `ctags`), the directory you installed VIDE to needs to be on your environment PATH. (Your compiler must also be included in the PATH.) To do this on Windows 98, you must edit the file `C:\AUTOEXEC.BAT`. You should find a line in it that begins with "PATH=". If you've installed VIDE in the default location, you can add "C:\PROGRA~1\VIDE" to the path (note the "PROGRA~1" instead of "PROGRAM FILES"), separated from other path directories by a semicolon. On NT and 2000, you use the system settings menu off the Start menu to change the PATH in the environment.
- **Set VIDE options** It is **essential** that you properly set some VIDE options to make it work correctly with the Borland compiler. Most options are found from the "Options" menu after you start VIDE. After you set these options, you must *exit from VIDE and then run it again* in order for them to take effect.
 - ◆ The dialog opened from the Options→VIDE menu contains the most critical options you must set. First, check the radio button to select "Borland" as the compiler. Then fill in the "Compiler root" box with the path to the Borland root directory. If you installed the Borland compiler to the default location, this will be "C:\borland\bcc55" (without the final \bin!). Finally, set the debugger to "td32".
 - ◆ You can use Options→ to set various attributes for the editor, including command set, indentation, and syntax coloring.
 - ◆ The Options→Font menu allows you to set the monospaced font of your choice. If you ever find that the cursor in VIDE does not work properly, it usually means you've re-installed VIDE or altered the font set on your machine. Simply reselect your font if this ever happens.
 - ◆ You can make VIDE the default editor for your source files by using the Windows command: **Start**→**Settings**→**Folder Options**→**File Types** dialog to associate your source file types

(e.g., .c, .h, .cpp) with VIDE.

Your First Project for Windows with Borland BCC [top](#)

VIDE can be used simply as an editor to edit any text file such as a C++ program or an HTML file. Simply use the File menu to open a file of your choice. For a programming project, however, VIDE lets you create a project that makes working with your program much easier. The following steps describe how to create a project, and then how to compile your program.

VIDE has been designed to use the standard tools that come with the Borland compiler. Before compiling your program, VIDE will use the project options you've set to generate a Makefile to be used with the standard Borland make program. VIDE then runs Borland make to build your project. For most cases, you won't need to know the details of how VIDE generates the makefile. The makefile generated by VIDE does use separate runs of BCC32 and ILINK32 which is different than using BCC32 alone from a command prompt.

- **Create a New Project**

The first step is to create a new project by using the **Project->New C++ Project** menu command. You will be prompted for the name of your project. Generally, you will create your project file in the same directory that contains your source code.

Once you have selected the project name, you will see a dialog that lets you define important characteristics of your project. The options you set in this dialog determine some important default options used to generate the Makefile. First, select if you are generating a C or C++ program. You can mix C and C++, but you would have a C++ project. Fill in the Target name box. This is the name of the executable or library file you want to create, including an appropriate extension (such as .exe).

Next, select the type of application you are building. Your choices include a "Console Application", which is one that does not use the Windows GUI API, such as the standard "Hello, World" app. If you are going to build a Windows GUI application, select "GUI Application". This allows you to use the standard Win32 API. Note that the free Borland compiler does not include any advanced GUI libraries such as MFC. If you are using the V GUI library or OpenGL, check the appropriate box.

Your other options in the New Project dialog box include if you want to generate a Release version or a Debug version. (VIDE does not include an automatic option for switching back and forth between Release and Debug. You can either create two projects, or manually change the compiler switch from -v- to -v.) Finally, select the compiler. If you have properly installed VIDE for the Borland compiler, Borland will be the default. If Borland is not checked, please go over the installation instructions again.

- **Use Project Editor to Add Files**

After you've made the appropriate selections in the New Project dialog, the new project will be opened in the Project Editor. This lets you edit all the properties of your project. You can come back to the Project Editor at any time from the **Project->Edit** menu.

Typically, the first thing you should do is add the source files included in your project. Select the Files tab of the Project Editor, and add all the source files used by your project. (You only add the C/C++ source files, and not the .h header files.) If your GUI project has any resource files (.rc), include them, too.

If you don't yet have a source file, you can skip this step for now, and come back later to add them by re-opening the Project Editor with the **Project->Edit** command.

- **Set Paths**

If your project uses any libraries or include files that aren't on the standard compiler search path, pick the Paths tab to add these paths. Any paths you include from this dialog will be added to the makefile. You can also keep your source, object, and executable files in different directories if you want, although that is usually not necessary. It is easiest to keep your project and program files in the same directory, usually the current directory (.). Note that VIDE will automatically include the paths to the Borland compiler directory. These values will override any you might have set in the Borland .cfg files, but are the standard values you will usually need.

- **Set Defines**

If you need to pass any defines to the compiler, pick the Defines tab. First, add defines to the pool list (in the form -DFOO or -UFIE), then add those to the active list. Many programs will not need to use this feature. VIDE automatically includes the definitions needed to build executables compatible with Windows 9x, NT, and 2000. If you want Windows 2000 compatible executables, you will need to change the default definitions.

- **Add Libraries**

If your program uses any non-standard libraries, select the Files tab and add them to the Linker Flags box. (This line will already have some libraries specified.) For example, you could use the common controls library by adding "-lcomctl32" to the Linker Flags line. If you get a lot of undefined symbols when you compile your program, it is likely that you need to include some library on this line. It is up to you to understand which libraries your program needs.

- **Set Compiler Options**

Many programs will compile and run correctly using the default compiler settings. However, the compiler has many options that you may want to or need to use. Add any switches the compiler needs to the Compiler line of the Files tab dialog. See Borland's documentation (or the VIDE [Borland](#) reference) for details of these switches.

- **Select the Runtime library**

Your program requires a runtime library to run. The Borland compiler offers you several choices for the runtime library you use. By default, VIDE will use the appropriate static runtime libraries for a

console or GUI app (which is the same as the default the BCC32 command line interface uses). If you want to use the Borland DLL versions, you can select the Advanced tab to change these. More details can be found in the VIDE [Borland](#) reference.

- **Close the Project Editor**

Once you've add all the source files, and set up whatever other options you need to, close the Project Editor. You can edit the Project later from the **Project**→**Edit** tab. When you next run VIDE, you will need to re-open your project using the **Project**→**Open** command.

- **Edit your files**

After you have created your project, you can use the VIDE editor to edit any of them as needed.

- **Build your project**

Once you've defined your project and have all the source files edited and added to the project, you can compile your program. You can use the "build" button on the tool bar, or select an option from the **Build** menu. Remember that VIDE generates a standard Borland makefile, and runs Borland make to do the actual building of your project.

The results of the build are shown in the VIDE message window. If there are syntax errors in your source files, you can simply right click on the error message with the line number to open that file and go directly to that line. Fix your syntax error, and try to build again.

At this point, you've now used the basic features of VIDE. You can read about other VIDE features, such as using the debugger, in the [VIDE User Guide](#).

VIDE for Linux with gcc

Installing VIDE for Linux with gcc [top](#)

- Install the compiler VIDE is designed to work with the normal gcc compiler installed with most Linux systems, and you should not need to change anything to use VIDE on a Linux system.
- Install VIDE executable The Linux version of VIDE is distributed as a statically linked binary version for recent versions of Linux. It is based on the Motif version of V, and is statically linked to the MetroLink Motif library. To install, unzip and untar the distribution. You can install the binary almost anywhere you want, either on a system path such as /usr/bin or /usr/local/bin, or on your personal /bin directory.

The Linux distribution of VIDE does not include ctags, which is normally already installed on Linux development machines.

- Set environment PATH The VIDE directory should be on your PATH.
- Set VIDE options VIDE has several options that you might want to set to make it work best for you. Most options are found from the "Options" menu after you start VIDE.
 - ◆ The default values from Options->VIDE dialog are generally fine for gcc.
 - ◆ You can use Options-> to set various attributes for the editor, including command set, indentation, and syntax coloring.
 - ◆ The Options->Font menu allows you to set the font of your choice. Be sure to select a monospaced font or VIDE will not update the edit windows correctly.

Your First Project for Linux with gcc [top](#)

VIDE can be used simply as an editor to edit any text file such as a C++ program or an HTML file. Simply use the File menu to open a file of your choice. For a programming project, however, VIDE lets you create a project that makes working with your program much easier. The following steps describe how to create a project, and then how to compile your program.

VIDE has been designed to use the standard tools that come with your compiler. Before compiling your program, VIDE will use the project options you've set to generate a Makefile to be used with the standard gnu make program. VIDE then runs make to compile your project. For most cases, you won't need to know the details of how VIDE generates the makefile.

- ### Create a New Project

The first step is to create a new project by using the **Project->New C++ Project** menu command. You will be prompted for the name of your project. Generally, you will create your project file in the

same directory that contains your source code.

Once you have selected the project name, you will see a dialog that lets you define important characteristics of your project. The options you set in this dialog determine some important default options used to generate the Makefile. First, select if you are generating a C or C++ program. You can mix C and C++, but you would have a C++ project. Fill in the Target name box. This is the name of the executable or library file you want to create, including an appropriate extension (such as .a).

The current version of VIDE only provides automatic support for V GUI projects. If you are using the Athena or Open Motif version of V, then select one of the two. If you select one of these two, then the Linker flag box will be filled in with appropriate options. Otherwise, you will find the Linker Flags box empty. It is up to you to fill in the appropriate switches for the library you are going to use.

Note that the X version of VIDE and V assume the use of the new, free version of Open Motif. Since this is a free, or nearly free version, of Motif, V and VIDE will no longer be tested with LessTif or commercial Motif versions. In fact, the Open Motif version seems to work wonderfully with V – the best Motif version ever. And it even gets the default decoration colors right under Gnome!

Your other options in the New Project dialog box include if you want to generate a Release version or a Debug version. (VIDE does not include an automatic option for switching back and forth between Release and Debug. You can either create two projects, or manually change the compiler switch from `-g` to `-O`.)

- **Use Project Editor to Add Files**

After you've made the appropriate selections in the New Project dialog, the new project will be opened in the Project Editor. This lets you edit all the properties of your project. You can come back to the Project Editor at any time from the **Project**→**Edit** menu.

Typically, the first thing you should do is add the source files included in your project. Select the Files tab of the Project Editor, and add all the source files used by your project. (You only add the C/C++ source files, and not the .h header files.)

If you don't yet have a source file, you can skip this step for now, and come back later to add them by re-opening the Project Editor with the **Project**→**Edit** command.

- **Set Paths**

If your project uses any libraries or include files that aren't on the standard compiler search path, pick the Paths tab to add these paths. This might include the path to whatever X library you are using (Motif, Athena, gtk, etc.). Any paths you include from this dialog will be added to the makefile. You can also keep your source, object, and executable files in different directories if you want, although that is usually not necessary. It is easiest to keep your project and program files in the same directory, usually the current directory (.).

-

Set Defines

If you need to pass any defines to the compiler, pick the Defines tab. First, add defines to the pool list (in the form `-DFOO` or `-UFIE`), then add those to the active list. Many programs will not need to use this feature.

-

Add Libraries

If your program uses any non-standard (non-C runtime) libraries (such as your X GUI library), select the Files tab and add them to the Linker Flags box. For example, you could use the Motif library by adding `-lXm` to the Linker Flags line. (Using Motif usually requires several other libraries as well. Consult the documentation for whatever X library you are using.) If you get a lot of undefined symbols when you compile your program, it is likely that you need to include some library on this line. It is up to you to understand which libraries your program needs.

-

Set Compiler Options

Many programs will compile and run correctly using the default compiler settings. However, the compiler has many options that you may want to or need to use. Add any switches the compiler needs to the Compiler line of the Files tab dialog.

-

Close the Project Editor

Once you've added all the source files, and set up whatever other options you need to, close the Project Editor. You can edit the Project later from the **Project**→**Edit** tab. When you next run VIDE, you will need to re-open your project using the **Project**→**Open** command.

-

Edit your files

After you have created your project, you can use the VIDE editor to edit any of them as needed.

-

Build your project

Once you've defined your project and have all the source files edited and added to the project, you can compile your program. You can use the "build" button on the tool bar, or select an option from the **Build** menu. Remember that VIDE generates a standard gnu makefile, and runs gnu make to do the actual building of your project.

The results of the build are shown in the VIDE message window. If there are syntax errors in your source files, you can simply right click on the error message with the line number to open that file and go directly to that line. Fix your syntax error, and try to build again.

At this point, you've now used the basic features of VIDE. You can read about other VIDE features, such as using the debugger, in the [VIDE User Guide](#).

Installing VIDE for Sun Java (JDK) [top](#)

Please see the complete [VIDE Java tutorial](#).

No Warranty [top](#)

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler
All rights reserved.

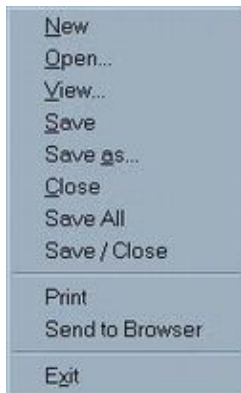
Bruce E. Wampler, Ph.D.
bruce@objectcentral.com
www.objectcentral.com

VIDE – Command Reference

This section provides a summary of the all the VIDE commands available from the menus.

- [File Menu](#)
 - [Edit Menu](#)
 - [Project Menu](#)
 - [Build Menu](#)
 - [Tools Menu](#)
 - [Options Menu](#)
 - [Help Menu](#)
 - [Debug Dialog](#)
 - [Warranty](#)
-

File Menu [top](#)



The File menu is used for source files. Use the Project menu to open and edit project files.

File:New

Create a new source file. Syntax highlighting doesn't take effect until you've done a **File:Save as** and repainted the screen.

File:Open...

Open an existing file.

File:View...

Open an existing file for viewing. The file is read-only, and you won't be able to make any changes.

File:Save

Save the current file to disk.

File:Save as...

Save the current file using a new name you will specify.

File:Close

This will close the existing file. If you've made any changes, you will be prompted if you want to save them.

File:Save All

Saves all currently open files.

File:Save/Close

Save the current file, then close it.

File:Print

This will print send a copy of the file in the active window to the printer. So far, there is no significant formatting. There is a simple header with the file name, date, and page number on each page. The code is printed in a 9 point fixed font with about 70 lines per page. Eventually VIDE may support syntax highlighting on the printed page.

File:Send to Browser

Save the current file, then open it with default browser. This command has a quick and dirty implementation, and it passes just the default file name to the system routine that opens the browser. Thus, this command can fail if the file doesn't have a full path qualification. This can happen when you type a file name in directly to the file open dialog. On MS-Windows, you must have .htm and .html files associated properly with your browser. This command won't do anything for non-HTML files. This association will usually be set automatically when you install your browser.

File:Exit

Exit from VIDE.

Edit Menu [top](#)

The edit menu has some basic commands to edit text in the current file.

Edit:Undo

Restores the last text deleted. Only one level. Doesn't undo insertions or position changes. Also, doesn't undo deletions greater than 8K characters.

Edit:Cut

Delete the highlighted text, and copy it to the clipboard. Standard GUI operation – use mouse to highlight region of text, then cut

Edit:Copy

Copy highlighted text to clipboard.

Edit:Paste

Paste the text on the clipboard to the current text position.

Edit:Find...

Find a text pattern in the file. Brings up a dialog.

Edit:Find Next

Find the next occurrence of the current pattern in the file.

Edit:Replace...

Find a pattern in the file, and replace it with a new one. Brings up a dialog.

Edit:Find Matching Paren

If the cursor is over a paren character, i.e., ()[] {}, the cursor will be moved to the matching opposite paren.

Edit:Set BP

This command will set (or preset if the debugger isn't running) a breakpoint on the current line. Breakpoints will be highlighted in yellow. Usually, you set breakpoints after you run the debugger, but VIDE remembers breakpoints across debugger sessions. (However, VIDE does not remember breakpoints across VIDE sessions!)

Edit>Delete BP

This command will delete the breakpoint on the current line.

Edit:Edit Help

Displays a list of the command supported by the current editor command set.

Project Menu [top](#)**Project:Open** 

Open an existing project file. Project files all have a `.vpj` extension. VIDE automatically detects C/C++ or Java projects.

Project:New C++ Project

This will create a new C/C++ project. The options on the dialog are described in the [C/C++ section](#) of this documentation.

Project:New Java Project

This will create a new Java project. The submenu allows you to create a new Applet, Windowed App, or a Console App. The details are described in the [Java](#) section of this document.

Project:Edit

Edit the currently open project. See [C/C++](#) or [Java](#) sections for details.

Project:Close

Save and close the currently open project.

Project:Save as...

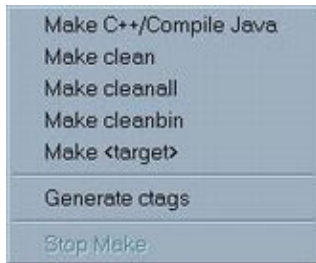
Save the current project under a new name. This is useful for creating "template" projects that have specific settings for your development environment. These templates can be opened later, then saved under a new name again for the real project.

Project:Rebuild Makefile

OK, I admit it. VIDE doesn't handle all cases of changes to your files. If you add a new `#include` to a source file, for example, VIDE won't automatically rebuild the Makefile to add this new dependency. This command helps get around that problem.

Project:Select Makefile or Java file

Instead of using a VIDE project, you can simply use an existing Makefile, or even a Java source file. Use this menu item to specify the Makefile or Java source file instead of a VIDE project. When you have a Makefile or Java source file selected, the Makefile will be run, or the Java source passed to the Java compiler when you click the make tool bar button.

Build Menu [top](#)

Used to build and compile projects.

Build:Make C++/Compile Java 

Build the project. This command first saves all your open files. It then runs `make` for C/C++ projects, or the Java compiler for Java projects. Errors are displayed in the message window, and you can go directly to the error by right-clicking on the error line in the message window.

Build:Make clean**Build:Make cleanall****Build:Make cleanbin**

Runs `make` with the given target: `clean` to clean object files, `cleanall` to clean objects and binaries, and `cleanbin` to clean binaries only. Used only for C/C++ projects.

Build:Make <Target>

Runs `make <target>` to make the target you specify. Used only for C/C++ projects.

Build:Generate ctags

Use this command to generate or regenerate the `ctags` file for the current directory. See [editor ctags](#) for more information.

Build:Stop Make

For C/C++ makes, will stop the make after the current file is finished being compiled.

Tools Menu [top](#)



Tools:Run project

Runs the program from the existing project. Note: VIDE does not check to recompile before running an object.

Tools:Run program w/ args

Runs a specified program. Allows you to specify arguments to the program.

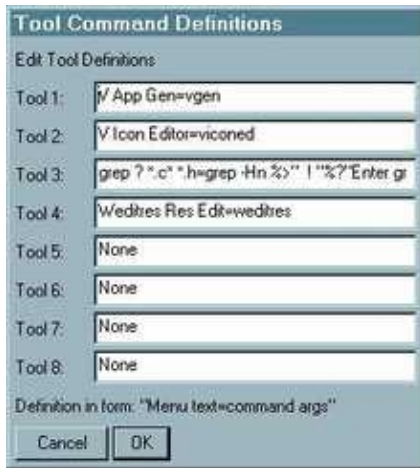
Tools:Start Debugger

For gcc versions using gdb, this command opens a new command window to interface to the debugger with the current executable file. When used with Borland TD32, it will launch Turbo Debugger with the current executable.

Tools:Run OS Shell

Runs a standard OS shell in a console window.

Tools:Setup Tools



This command lets you define and setup the remaining eight tools shown on the menu. The first few are predefined by VIDE when it is installed, but you can change them if you have other tools more important to you. When you click on **Setup Tools**, you will get a dialog box with a definition of 8 tools. The definitions of the tools must be in the format: "Tool Menu Description=command args". The first part of the definition is what will be shown on the Tools menu. The part after the "=" defines how the tool will be run. VIDE supports several parameters that can define various substitutions and characteristics used to run the command.

For example, the `grep` tool is defined as:

```
grep ? *.c* *.h=grep -Hn %*%>\" | \"%?\"Enter grep search pattern\"%_ *.c* *.h
```

The `grep ? *.c* *.h` is what is displayed on the Tools menu, and in this case indicates `grep` will be run for `*.cpp` and `*.h` files. The part after the "=" defines how the `grep` command will be run, with "%" commands interpreted by VIDE before `grep` is actually run. The first part, of course, is `grep -Hn`, which is the name of the command with some standard option arguments (`-Hn`). You can include a full path here if you need to, or simply let the standard environment `PATH` be used to run the command. In this case, VIDE relies on the fact that the `grep` executable will reside somewhere on the standard system `PATH`.

The end of the definition, `*.c* *.h` defines the file names passed to `grep`. The stuff in the middle with the "%" commands is interpreted by VIDE. The first command, "%*" tells VIDE to save all open files before running the tool. Thus `grep` will search all the latest versions of you files. The next, "%>", means that the output of the command (`grep`) is to be redirected to the VIDE message window. The string after the "%>", " |" is used to set a prefix that will be used in front of the `grep` output redirected to the message window. The "%?" parameter is used to prompt the user for input, in this case the `grep` search pattern, with the supplied message. The "%_" means that `grep` should be run minimized, which means it won't flash on the screen. By using a combination of the different % parameters supported, you can run external tool commands with many options. The description of the % parameters and commands supported by VIDE follow. Note that in some cases, the % command will be replaced in the definition with a file or path name, while in others, the % parameter is removed from the command before it is run.

- %T – Substitute the current target defined by the project. This will usually be the executable program name.
- %F – Substitute the name of the file in the currently active window. This will be the window that the tool is run from.
- %B – Substitute the current binary directory as defined by the current project. Note that this will not include the trailing / or \.

VIDE

- %R – Substitute the name of the file in the currently active window, but with the ".ext" stripped. This will be the window that the tool is run from.
- %S – The current project source directory.
- %O – The current project object directory.
- %/ – Convert files or directory separators to all '/'s.
- %\ – Convert files or directory separators to all '\'s (for Windows).
- %>["prefix"] – Redirect tool output to the VIDE message window (this automatically implies the % command as well.) If a string is provided right after the >, show the output of the tool prefixed with the supplied string in the message window. The default prefix is ">".
- %! – Save contents of the current file, run the tool, then reload file. This is useful if the tool might modify the contents of the currently open file.
- %?["prompt"] – Prompt user for arguments for tool. The user input replaces the %?. If a "prompt" provided, user will be prompted with that string if no "prompt", then a standard prompt used.
- % . – Wait for the tool to complete before continuing with VIDE.
- % _ – Start the tool minimized – useful to avoid a flash.
- %% = Insert a %.
- %* – This will cause all open files to be saved before running the tool. Useful so the tool will work on the latest versions of your files.
- %: – Run tool in a console. This is sometimes necessary for Windows NT and 2000. (It also behaves slightly differently for Linux.) If you try to define a tool and all you see is a quick flash of a window, try adding %: and the tool should then run.

The remainder of the commands on the Tools menu can be defined or redefined using the Setup Tools command. The following tool definitions are provided by default:

Tools:V App Gen

Runs the V tool V App Gen.

Tools:V Icon Editor

Runs the V Icon Editor.

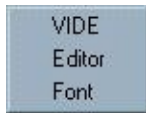
grep ? *.c* *.h

Runs the standard grep program that will search all *.c* and *.h files in the current directory for a regular expression (pattern) entered by the user in response to a prompt. The results are directed to the VIDE message window, and you will be able to right click on a line to go to that line in the file.

This is a *very* useful feature! The VIDE distribution contains an HTML [grep](#) help file for details on forming regular expressions.

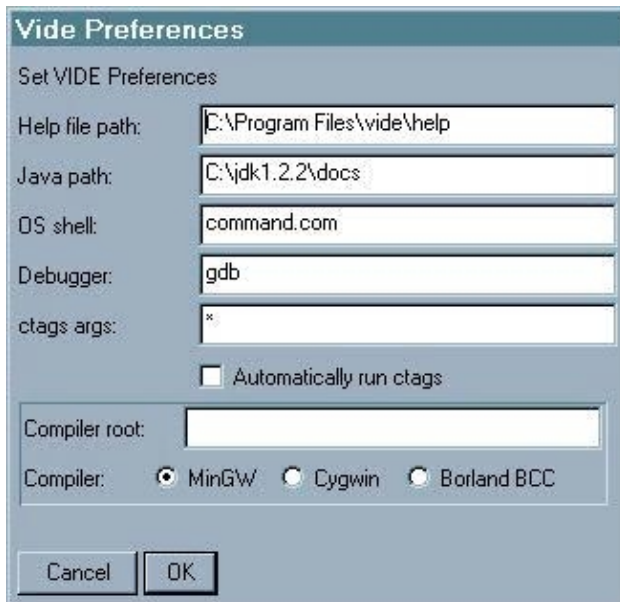
Here is a very useful example for using the grep tool. Suppose you have several .html files in one directory that contain the string "1.24" and you want to find all the occurrences and maybe change them to "1.24" depending on the context. You can use the grep tool to find each file with the string and then right click in the VIDE message window to selectively open each file with the string. You would enter the command Tools->grep, enter the pattern "1.24" to the prompt. (Note that you can add other file extensions to search easily at the prompt.) Then grep would run and put all the matches with file names and line numbers in the VIDE message window.

Options Menu [top](#)



The Options menu allows you to customize various aspects of VIDE, including paths, editor attributes, and font. These settings are saved in a standard system place. For example, they are saved in `C:/windows/videl.ini` on MS-Windows. They will be in `$(HOME)/.Viderc` on Linux or other Unix-like systems.

Options:VIDE



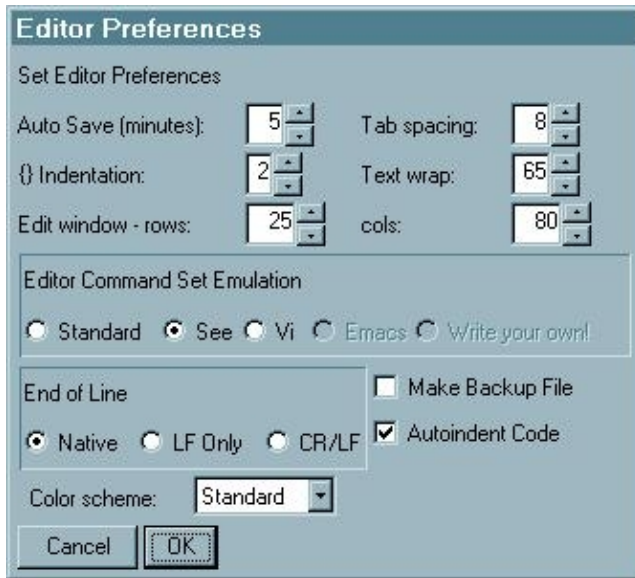
This item allows you to set the paths VIDE uses to find the VIDE help system files as well as the standard help file included with the Sun JDK.

This item also allows you to specify which command shell and debugger are used. These two options are most useful on Linux/Unix systems.

You can also set the default args used when running ctags. See [editor ctags](#). If you set the "Automatically run ctags" box, then VIDE will automatically generate a new ctags file whenever you open a project. Note that running ctags is a very fast operation.

For the MS-Windows version of VIDE, you should also select which compiler you are using. The currently supported compilers include MinGW gcc, Cygwin gcc, and Borland's BCC 5.5. The **Compiler root** setting is currently used only for support of the Borland C++ Compiler 5.5. See the [Borland reference](#).

Options:Editor



The options include:

- **AutoSave** – VIDE will automatically save changed versions of edited files every N minutes. Use 0 for no autosave. Note: while autosave might save a lot of lost work for you sometime, it can have problems, too. Unless you check the "Make Backup File" option, VIDE only keeps the current copy of your file. After you have saved the file, either by a manual save, or by autosave, previous versions of your file are lost. If you quit the editor, and answer no to saving changes, you won't necessarily get back to your original file if you have autosave on.
- **Tab Spacing** – the number of spaces for each TAB character. The most widely used value is 8.
- **{ } Indentation** – This applies to beautifying Java and C++ code. With a value of 0, braces { and } will be lined up with the outer indentation level. With a value of 2, braces are indented 2 from the outer level. It is most common to line up with the outer level (value 0) in code that lines up braces, but I find the extra 2 spaces makes the braces stand out better, and is easier to read. This should be 0 for KRstyle code.
- **Text wrap** – If you are editing text or HTML files, each editor emulation will provide a command that will automatically fill text. The text wrap value is the column used to determine the wrapping.
- **Editor emulation** – Select which editor command set you want to use.
 - ◆ Standard Editor
 - ◆ See
 - ◆ Vi

The editor emulation changes for new edit windows you open. The editor you chose will be saved in the preferences file.

- **Make backup file** – if this is checked, VIDE will make a backup version of the original file. The backup will retain the original name and extension, but add a ".bak" to the name. For example, "foo.cpp" will be saved as "foo.cpp.bak". This feature can be handy when used in conjunction with AutoSave to be sure there is an unchanged copy of the original file.
- **Autoindent Code** – When this is checked, VIDE will autoindent when you are entering new code. When you enter a newline, VIDE will automatically insert the same indentation as the previous line (spaces and tabs). This will be done only for code files (C++, Perl, Java).
- **End of Line** – This controls how VIDE saves the end of line in your file. Windows uses a CR/LF by default, while Linux/Unix systems use LF. This option is most useful on Windows to save the files with a LF only. Note that the gcc compiler prefers files with LF only, but will correctly use CR/LF files.

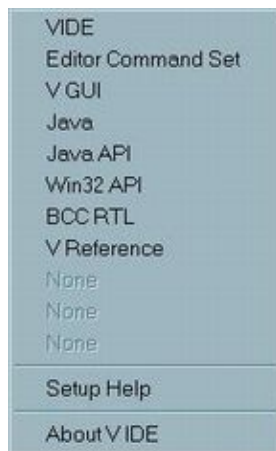
VIDE

- **Color scheme** – By popular request, VIDE now includes several color schemes for the editor windows. The standard version is black letters on a white background, with various other colors used for syntax highlighting. The color scheme option lets you pick from several other color schemes. You can only pick from the fixed schemes. If you have a color combination you'd really like to see, send e-mail and I will consider adding it for the next version. When you select a color scheme, it is not applied until you open a new editor window.

Options:Font

Lets you specify the font used in the display window. The font will change in the current window, and in future windows, but not in already open windows. The font you chose will be saved in the preferences file.

Help Menu [top](#)



The Help menu will open help files specified for your copy of Vide with the **Setup Help** command. The help files are often found in a directory specified in the "Help file path" and "Java path" from the **Option->VIDE** menu. Several default help files are provided, but you may specify any 10 help files you want.

Help:VIDE

Opens your browser with this file. Uses file `videpath/videdoc/videdoc.htm`.

Help:Editor Command Set

Shows a dialog box with a command summary of the editor command set currently being used. This is the only help item that doesn't use a file.

Help:WIN32 API

This will try to open the Borland WIN32 API .hlp file. The hlp format file should be available at [Windows API Reference](#) from Borland. Uses file `videhelp/win32/win32.hlp` first, then tries `videhelp/win32/win32.htm`.

Help:V GUI

Opens the V GUI Reference Manual. Uses file videhelp/vrefman/v.htm.

Help:Java JDK

Opens the top level Sun JDK Help pages. You must set the Java help path in the Options:Editor menu, and download the help files from Sun. Uses file javapath/index.html.

Help:Java API

Opens the Sun JDK API Help pages. These cover all the standard elements and library classes of Java, and is probably the reference you will most often use. You must set the Java help path in the Options:Editor menu, and download the help files from Sun.

Help:BCC RTL

Opens a guide to the Borland BCC runtime library. Uses file C:\Borland\bcc55\Help\Bcb5rtl.hlp.

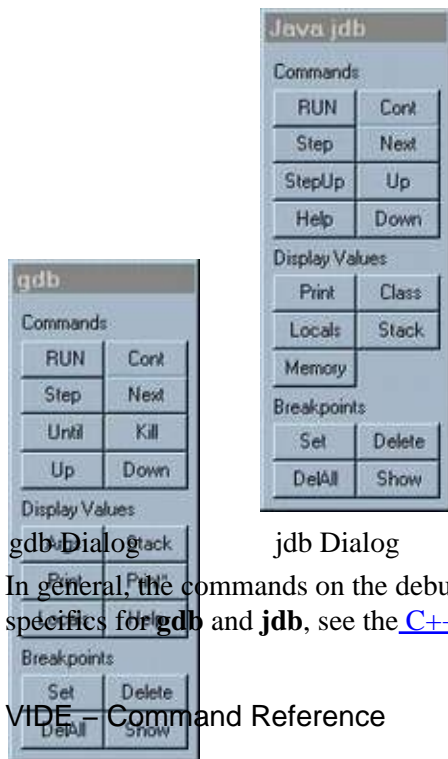
Help:Setup Help

Allows you to define or redefine up to 10 help files.

Window Menu

Standard MS-Windows Window menu.

Debug Dialog [top](#)



In general, the commands on the debug dialog work very similarly for the **gdb** and **jdb** debuggers. For more specifics for **gdb** and **jdb**, see the [C++ Tutorial](#) or the [Java Tutorial](#).

Commands:RUN

Run the program being debugged from the start. You will usually want to set some breakpoints first.

Commands:Cont

Continue running program until the next breakpoint is reached.

Commands:Step

Step into the current statement. This will break at the first statement of a called function. For non–function calls, this will have the same effect as the Next command. When you use Step or Next, the new current program line will be highlighted in red.

Commands:Next

Step over the current statement. If the statement is a function call, the program will break after the function has returned.

Commands:Until

Continue running the program until the current line in the editor window is reached. (gdb only)

Commands:Kill

Stop execution of the running program. (gdb only)

Commands:Up

Move execution up stack frame.

Commands:Down

Move execution down stack frame.

Commands:StepUp

Execute until the current method returns to its caller. (jdb only)

Display Values:Args

Display args to current function. (gdb only)

Display Values:Stack

Display the call stack.

Display Values:Print

Display the value of the variable highlighted in the editor window. The variable must be available in the current context of the running program. To use this command, use the mouse to highlight the variable name

you want to inspect. It is easiest to double click over the symbol to highlight it. Then click this command in the dialog.

Display Values:Print*

Line Print, but does indirection. Useful for C/C++ pointers. (gdb only)

Display Values:Locals

Print all local variables in current stack frame.

Display Values:Class

List currently known classes. (jdb only)

Display Values:Memory

Report memory usage. (jdb only)

Breakpoints:Set

Set a breakpoint at the current line in the editor window. To use this (and other commands that use the "current line"), first get focus to the editor window of the source file you want to work with. Then go to the line you want to work with, either with the mouse or cursor movement commands. Finally, click the *Set* button in the dialog box (or the Edit:Set DB menu). Breakpoints will be highlighted in yellow. Usually, you set breakpoints after you run the debugger, but VIDE remembers breakpoints across debugger sessions. (However, VIDE does not remember breakpoints across VIDE sessions!) When you hit a breakpoint, the current program line will be highlighted in red.

Breakpoints>Delete

Delete the breakpoint set at the current line. This command isn't as easy to use as it could be because the editor doesn't highlight lines with breakpoints. The current version doesn't keep its own list of breakpoints, but relies on the debugger. Thus, you have to know in advance which line you want to debug. It is sometimes easier to just use the debugger command line interface for this. I hope this one gets better in the future.

Breakpoints:DelAll

Delete all set breakpoints.

Breakpoints:Show

Show all set breakpoints. Uses native debugger commands for this.

(Breakpoints,Display Values):Help

Show debugger help. Uses debugger native help command.

No Warranty [top](#)

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler
All rights reserved.

Bruce E. Wampler, Ph.D.
bruce@objectcentral.com
www.objectcentral.com

VIDE – Editor Reference

- [The VIDE Editor](#)
 - [Getting started – Tips](#)
 - ◆ [Everybody Does It!](#) – General Editing
 - ◆ [Do It Your Way!](#) – Emulations
 - ◆ [Do It To What?](#) – Selecting text
 - ◆ [Where Is It?](#) – Finding text and symbols
 - ◆ [Move It!](#) – Moving text
 - ◆ [Make It Pretty!](#) – Looking good
 - ◆ [Do It Again!](#) – Macros
 - ◆ [I Did It Wrong!](#) – Undoing
 - [VIDE Features](#)
 - ◆ [Keyboard Macros](#)
 - ◆ [Code Beautifier](#)
 - ◆ [Syntax Highlighting](#)
 - ◆ [ctags](#)
 - ◆ [Other Features](#)
 - Editor command summaries
 - ◆ [Common to all emulations](#)
 - ◆ [Standard Editor](#)
 - ◆ [See](#)
 - ◆ [Vi](#)
-

The VIDE Editor

What is the most important part of any IDE? Why, the editor, of course! No matter how easy an IDE makes it to compile and debug your programs, you will still spend most of your time in the editor. The VIDE editor is based on the V TextEditor class, and has a long history.

I wrote the first version of the editor over twenty years ago in a language called Ratfor. Over the years, this editor code has evolved. It went from Ratfor to C to its current C++ version, and from supporting a simple console terminal to the current GUI version in an IDE. Over all this time, I always have found it easier to port the editor rather than learn a new one. This editor has the features that I've found the most useful as a programmer. For most of its life, the editor supported my own command interface called See, but the current version supports a standard GUI interface, as well as a Vi command emulation. No matter which command interface you use, the VIDE editor has a long history and includes many features very useful to a programmer.

Getting Started – Tips [top](#)

As I just noted, the VIDE editor has evolved into a fully featured programmer's editor. But in order for you to get the full value of these features, you have to know about them. This section will help you get started using the VIDE editor. It contains some tips for getting the most out of the VIDE editor — understanding both its features, and its quirks. Do yourself a favor, and read this section!

- **Everybody Does It!** – General Editing

Of course the editor supports all the standard things you usually do with an editor – moving the cursor, adding text, changing text, finding text, and so on. Just how you do these actions depends on the editor command emulation you are using. (See next section.) VIDE has a very extensive help system, but there is an extra menu command that you will likely find useful as you are first learning the editor. Using the **Edit→Editor Help** menu command will bring up a dialog box with a list of the commands supported by the current editor emulation.

- **Do It Your Way!** – Emulations

The current version of VIDE supports several command sets. The default is a generic modeless command set typical of most GUI editors. The second command set is based on the See editor which is the editor I've been using for 20 years. It is not well known – yet. There is also a very good emulation for Vi that will keep your fingers happy if you use Vi. It is possible to add a new emulation with just several hours of work. Adding an emacs emulation has been on my list for a long time, but I've yet to receive even one request to do it. The main idea of any emulation is to support "finger memory" – allowing your fingers to do what they have learned as you've used your favorite editor. Thus, I hope I've succeeded making the generic GUI editor like most other GUI based editors, and having the Vi emulation work as you would expect. You can select the emulation from the [Options→Editor](#) dialog.

Some features of the editor are available mainly through menus and mouse actions. For example, all emulations use the mouse to access the ctags features. Some of the searching and replace features are available only through the Edit menu.

- **Do It To What?** – Selecting text

GUI applications all allow you to highlight text for some kind of action – copy to the clipboard, delete, and so on. VIDE supports standard mouse based selection. You can also use Shift+<arrow-keys> to highlight selections. Because of limitations in the V library, you can use the mouse only to select currently showing text. You *can* use the Shift+:<arrow-keys> to select text off the display window.

You can left-click to select text. The first click positions the cursor. The second click will select the word (a–z, 0–9, and _) under the cursor. A third click will try to select a complete programming symbol (although the selection algorithm still needs some improvement). A fourth click will select the entire line. This selection method is very useful for looking up ctags and setting breakpoints.

- **Where Is It?** – Finding text and symbols

Of course, one of the main things an editor lets you do is find text within a file by using a *find* or *search* command. The VIDE editor supports this, but it also lets you easily find the definition of a program symbol using another program call *ctags*.

- ◆ *Searching*

The VIDE editor supports finding (and replacing) text two ways. There are three Edit menu options: Find, Find Next, and Replace. The menu forms use dialogs to enter the patterns. Each editor will also have keyboard commands that let you enter the patterns without a dialog. The patterns are echoed on the status line. VIDE does not support regular expressions

in its find patterns.

◆ *ctags*

The *ctags* program has been used for years on Unix systems. Essentially, *ctags* builds an extensive cross-reference of the symbols (variables, function names, defines, etc.) in your program files. The VIDE editor uses the *ctags* file to let you find the original declaration of any symbol in your source file. It is simple to use. Highlight the symbol you want to look up (double left-clicking over the symbol is the easiest way to do this). Then right click on the highlight symbol. The VIDE status window will then show locations where the symbol has been defined or declared. If you then want to see the actual definition, right click on the line in the status window, and the file will be opened right on that line. Neat!

• **Move It!** – Moving Text

Moving big blocks of text around is a common editing operation. The standard GUI way to do this is to cut and paste the selection. The See and Vi emulations also support other very powerful ways to move blocks of text, including reading and writing files. See the See "Save buffer commands" and the Vi "Yank buffer commands" for more details. Currently, the standard GUI editor does not support this concept.

• **Make It Pretty!** – Looking good

Having nice looking code is a *very* important part of the programming process. A well formatted, properly indented program makes it far easier to read, understand, and maintain. The VIDE has several features that make it easier for you to write good looking code. It even has some features to let you write documentation – in plain old text or HTML.

◆ *Auto-indent*

If you are editing code files, the VIDE editor has an auto-indent mode (enabled in the **Options**→**Editor** dialog). When auto-indent is on, the editor will automatically indent the same number of tabs and spaces on the previous line whenever you insert a new line. If you need to unindent, simply press the backspace key.

◆ *Beautifier*

I personally don't like auto-indent that much. Instead, I prefer to use the VIDE beautifier. This feature will automatically beautify (set the proper indentation) source code, and fill text. The neat thing is that you can do this any time, and it will clean up your code even after you've done extensive editing. A beautifier may not sound really important, but over the years, I've found that it is one of the most important features of the editor, and I would find it difficult to program without it. There are more details in the [Code Beautifier](#) section.

◆ *Syntax highlighting*

The VIDE editor knows how to highlight the syntax of C, C++, Java, Perl, and HTML. Syntax highlighting makes it much easier to read your code. There are several standard color choices available in the **Options**→**Editor** dialog.

◆ *HTML support*

VIDE is not a full blown HTML environment, but it does have a few features that help. I use it for most of my HTML editing. As a programmer, I find it pretty easy to use the standard HTML tags, and find that the editor I use (VIDE/See) is more important. In addition to HTML syntax highlighting, the **File**→**Send To Browser** command is most useful. It will

close the current file, and then use the default browser to open it. This is pretty effective. Once the current file has been loaded into the browser, it is often easier to click the save file tool bar button, and then the reload button in the browser. I've been doing that a lot as I write this, and it is almost as good as a full HTML editor.

• Do It Again! – Macros

There are a lot of editing tasks that are repetitive. VIDE has a couple of features that makes these tasks easier. First, most editor commands all you to add a count at the beginning. Thus, you can enter a command that says go down 53 lines instead of pressing the down arrow 53 times. The mechanism for entering counts depends on the command syntax of the given emulation.

Even more powerful than simple counts is the VIDE macro facility. Essentially, a macro is a small editing program that you can execute. You first define a macro consisting of editor commands needed to perform the task, and then execute the macro as many times as needed. See the [macros](#) section for more details.

• I Did It Wrong! – Undoing

You mean you make mistakes when editing? VIDE has a few features to help you undo editing errors.

- ◆ Undo
VIDE has an undo command, but it is only one level, which works for the most common situations. Essentially, you can undo the last delete operation, which most of the time will be enough. You can't undo a long insertion (do it by hand!), or a series of single character deletes. VIDE's undo facility is one place where the 20 year heritage of the code limits it functionality – the internal editor code just doesn't lend itself to multiple undos and redos.
- ◆ Autosave
Probably the biggest safety feature of VIDE is autosave. You set the value in the **Options**→**Editor** dialog. The current state of your file will be automatically saved at the frequency you specify. This is five minutes by default.
- ◆ Backup file You can also use the **Options**→**Editor** dialog to tell VIDE to make a backup copy of the original file when you open it. With this enabled, you will always have the original copy of the file to revert to in case of a major editing error.

VIDE features [top](#)

This section describes some of the features of the VIDE editor emulations that are especially useful for programming. While most of these features are found in other editors, some are not native to the original editors VIDE emulates. In those cases, VIDE tries to fit the features into the natural command syntax of the original editor.

VIDE Keyboard Macros [top](#)

The Standard, See, and Vi emulations all support a simple, yet powerful macro facility. There are 26 buffers called "Q-Registers". You can record character keystrokes into any of the Q-registers, and then execute those characters as a set of commands. A set of commands like this is often called a macro, or in VIDE terminology,

Q–Macros. (Note: when you record special keys such as the arrow keys, they are all echoed as "{fn}" — you won't be able to tell from the echo which key you entered.) You can enter any command into the editor, including find patterns and insertion strings.

The general procedure for using Q–Macros is the same on the currently supported emulations (the commands for using Q–Macros are different). First, you record a sequence of keystrokes into one of the 26 Q–Registers. The keystrokes you enter will be echoed on the status line. You can use backspace to edit your input. You terminate the Q–Macro by entering the special end–macro character defined for the emulation.

Once you have a macro recorded, you can then execute it using the execute–macro command of the emulation. You can provide a count, and thus execute the macro many times. Q–Macros terminate when any given command fails. For example, if a find fails, the macro will end. There are commands to specify which Q–Macro to execute.

Code Beautifier [top](#)

The VIDE editor has a code beautifier for C, C++, and Java. How the beautifier works for Perl has not been fully tested. The editor will also fill text lines for text files.

To use the beautifier, you place the cursor on the line after a line that is already properly indented. Then enter the beautify command (as defined by the given emulation), and your code will be automatically indented, subject to a few limitation described later. Some parameters of the filling and beautification process can be adjusted in the **Options**→**Editor** dialog.

Code Beautifier

VIDE will format your C, C++, and Java source code by following a fairly simple set of rules. Because VIDE doesn't fully parse the code, you will have to follow some coding conventions, and may have to "help" manually sometimes. On the whole, however, VIDE's beautify command works better than using auto-indent. It is especially useful after you've revised a section of code.

- Indentation is based on steps of *four* spaces. Following the normal Unix convention of 8 spaces per tab, your code will be indented to an even multiple of 4 spaces, even tab stops, or tab stops plus 4 spaces.
- The standard language keywords plus curly braces ({}) are used to determine indentation.
- Lines with non-whitespace in the first column are left as they are.
- Formatting is based on the indentation of the previous non-blank line. Thus, when you beautify your code, you must start at a place where the indentation is already correct.
- Because VIDE doesn't completely parse the source code, it doesn't always correctly handle `case` and `default` statements. You can usually work around this by manually formatting the first line of a case group or default group.
- Beginning with version 1.05, the VIDE beautifier can automatically handle two coding conventions. The preferred style assumes you have braces ({}) on lines by themselves, perhaps with a trailing comment. VIDE also tries formatting source code that uses the convention of placing the opening brace ({}) at the end of a control statement such as `if` or `while` (sometimes known as KRstyle). Support for KRstyle is not quite as robust as keeping braces on separate lines, but it is still good. Having a pair of braces on the same line can confuse VIDE's formatting.

Hint: when you are entering new code, it is often easier to not worry too much about the indentation. Just leave some whitespace at the beginning of each line, then go back and beautify it after you have your

statement structure complete.

Hint: Using a command count often is especially useful when beautifying blocks of code. Remember to start on a line that is indented the way you want already.

Text Filling

When you use the beautify command on text files, VIDE will fill the text to the column set in the **Options:Editor** dialog. If the first column of the text has certain special characters or character sequences, VIDE will skip filling that line. This feature is intended to make filling of HTML and other markup language files work better. VIDE won't fill if the line is blank, or begins with a space, a period, a tab, a latex keyword, or a block oriented HTML command.

Syntax Highlighting [top](#)

C, C++, Java, Perl

VIDE editors will highlight C, C++, Perl, and Java source code. The following default conventions are used: keywords in blue; constants in red; comments in green; C/C++ preprocessor directives in cyan; remainder in black. Other colors are used for alternate color schemes.

HTML

Highlighting of HTML files is very simple minded, but can really help you to read the HTML source code. Angle brackets (<, >) are always highlighted. If the first character sequence after the opening < is a valid HTML command, then it is also highlighted. Other parameter keywords within an HTML command are not highlighted. String constants and numbers are also highlighted. The "" character is also highlighted.

ctags [top](#)

Beginning with version 1.06, VIDE supports the *ctags* program. Ctags will generate a cross-reference tag file of C++ and Java source files. VIDE can read this file and locate the original definition or declaration of a symbol.

To use the ctags feature, first use the **Build→Generate ctags** command. This will generate a file called "tags" in the current directory. Then, to locate a symbol's definition, highlight it anywhere in a source file. (It is easiest to do this by double clicking over the symbol.) Any instances of that symbol will then be displayed in the status window. If the symbol is defined in multiple files, there will be multiple lines shown. There is extra information supplied about the symbol, and you can usually tell which is the instance you want. Often the information shown in the status window will be enough to help. If you need to see the actual definition or declaration, then right click the appropriate line in the status window, and the file will be opened.

Sometimes ctags will not include the symbol in the tags file. It does not generate entries for symbols local to a function, or for function parameters. And it will not automatically include symbols from libraries you use. If you want library symbols included in the tags file, you need to include the path to the library on the ctags args line.

By default, the arguments supplied to ctags are "-n", which is required for VIDE to use the tags file properly,

and "*", which will make ctags use all the source and header files in the current directory. Most of the time, this will be just what you need. However, ctags has many options, and can tag files from other directories given the proper options.

You can change the default "*" argument using the **Options**→**VIDE** dialog. You can also supply a project specific ctags argument list using the project editor. Note that the "-n" switch will always be used.

To use the ctags feature, you normally don't have to do anything other than be sure ctags is available. The Windows distribution of VIDE will install the ctags executable on the VIDE directory, and ctags is normally found on Linux systems. (You may have to add the VIDE directory to your AUTOEXEC.BAT PATH.) VIDE provides the ctags version known as [Exuberant Ctags](#). A local *text* copy of the [ctags man page](#) is also included with VIDE. The source of Exuberant Ctags is available at its web site.

Other features [top](#)

Wide lines

One attribute of the VIDE editor is how it handles lines wider than the window. It does not use a horizontal scroll bar. Instead, as you move right on long lines, the text will automatically shift to show more of the line. The last character displayed in the window of a wide line will be the last character in the line, and not the character that actually is in that column.

Auto save

You can set the Auto save value in the **Options:Editor** dialog to tell VIDE to automatically save open files at a given interval. This approach is different than some programmer's editors. You can end up with files that are in a state of transition. VIDE also supports making a backup file of the original when you edit. Note that whenever you do a project build, open files are automatically saved.

Time stamp

If you put a comment containing the string "date:" anywhere in the first 12 lines of your source code file (C++, Java, Perl, HTML), VIDE will automatically add a time stamp after the "date:" whenever you make changes to the file.

Commands common to all editors [top](#)

In an effort to comply with standard interface design, especially as it applies to MS-Windows, there are several commands that are common to all editor emulations. These common commands are explained in this section.

In addition to the following keyboard commands, the action of the mouse to move the cursor and select text conforms to normal interface design.

VIDE

Key	Command Description
	<i>Selection Highlighting</i>
<i>n</i> Shift–UpArrow	Extend selection <i>n</i> lines up.
<i>n</i> Shift–DownArrow	Extend selection <i>n</i> lines down.
Shift–LeftArrow	Extend selection left one character.
Shift–RightArrow	Extend selection right one character.
Shift–Home	Extend selection beginning of line.
Ctrl–Shift–Home	Extend selection beginning of file.
Shift–End	Extend selection end of line.
Ctrl–Shift–End	Extend selection end of file.
	<i>Clipboard</i>
^X [Menu Edit:Cut]	Cut selection to clipboard.
^C [Menu Edit:Copy]	Copy selection to clipboard.
^V [Menu Edit:Paste]	Paste clipboard to insertion point.
	<i>Menu Commands</i>
File:New	Create a new file. You will be prompted for the name of the new file.
File:Open	Open an existing file.
File:View	Open an existing file for read only access.
File:Save	Save (write) current file.
File:Save As	Save current file under a new name. New file becomes current file.
File:Close	Close current file.
File:Save All	Save all open files.
File:Save / Close	Save and close current file.
File:Send to Browser	Send current HTML file to browser. This is an effective way to edit HTML files and view the results.
Edit:Undo	VIDE's undo is a bit limited. Undo will undo the last text delete. It does not undo inserts, clipboard operations, or cursor movement.
Edit:Find	Opens the VIDE Find dialog. This dialog gives you all the options available for finding text. The emulations will support both dialog based finds, and command line finds.
Edit:Find next [F3]	Find next occurrence of find pattern.

Edit:Replace	This opens the Find and Replace dialog. If you check the Confirm Replace option, you will be prompted before the replace is done. That confirmation dialog will also allow you to go ahead and replace all or cancel the find and replace.
Edit:Find Matching Paren	Why is this a menu command? Because the command is useful for beginners who will often use menu commands over keyboard commands.
Edit:Editor Help	Brings up a dialog with a command summary for the editor emulation.

Note about selections. You can use the mouse to highlight an area of text as well as the keyboard commands listed above. If you need to select more text than shows in the window, you will have to use the key selection commands (e.g., Shift–Down).

The Standard Editor [top](#)

The generic command set of the VIDE editor is very similar to those found in many GUI based text editors. It is modeless. Commands are either special keys (e.g., arrow keys), or control keys (indicated by a ^ or Ctrl). Several commands use a meta modifier, Ctrl–A (^A) as a prefix. You can enter counts (noted by an "n") for commands by first pressing the Esc key, then a count, then the command. Not all commands are supported by command keys, and require you to use the menus (replace, for example). Note that the See and Vi editors have some features not found in the standard editor.

The Standard Command Set [top](#)

Key	Command Description
Esc	Prefix to enter count <i>n</i>
	<i>Movement Commands</i>
Arrow keys	Standard function
<i>n</i> Left	Move left [^L]
<i>n</i> Ctl–Left	Move left a word
<i>n</i> Up	Move up [^U]
<i>n</i> Right	Move right [^R]
<i>n</i> Ctl–Right	Move right a word
<i>n</i> Down	Move down [^D]
Home	Goto beg of line [^A,]
Ctrl–Home	Goto beg of file
End	Goto end of line [^A.]
Ctrl–End	Goto end of file
<i>n</i> PgUp	Move a screenful up
<i>n</i> Ctrl–PgUp	Scroll a screenful up
<i>n</i> PgDn	Move a screenful down

nCtrl-PgDn	Scroll a screenful down
	<i>Searching commands</i>
^A]	Balance match
^F	Find pattern (non-dialog). This form of find allows you to enter the find pattern directly from the keyboard. The find pattern is terminated with the Esc key. The pattern is echoed on the status bar. This form of find is useful for Q-Macros.
^A^F	Find pattern – use find dialog.
Shift-^F [F3]	Find next
	<i>Insertion Commands</i>
n^AIns	Insert char with value of n
n^O	Open a new blank line
Ins	Toggle insert/overtyp
	<i>Editing commands</i>
^ABkspc	Delete to line begin [^A]
^ADel	Delete to line end [^A]
nShft-^C	Fold case
^C	Copy highlight to clipboard
^V	Paste from clipboard
^X	Cut highlight to clipboard
nBkspc	Delete previous char
nDel	Delete next char
nShft-Del	Delete line
	<i>Macros</i>
^Aq<a-z	Set register to use (a-z)
^Q	Record keystrokes in Q-Register until ^Q
n^E	Execute current Q-Register N times
	<i>Misc. commands</i>
^AM	Center Cursor in screen
^Av	Repaint screen

$n^{\wedge}G$	Goto line n
$n^{\wedge}K$	Kill line
$n^{\wedge}B$	Beautify code. This beautifies code or fills text. The beautify command will format C/C++, and Java code according to the V style conventions. To use this command, start on a code line that is indented how you want it. After that, 'n' lines will be formatted based on the starting indentation. This command is very useful if you use the V indent style. You can set the indentation for braces in the Options:Editor dialog. For text files (including HTML), this command will fill lines to the column specified in the Options:Editor dialog.

The See Editor [top](#)

The command set of the See editor dates back to the late 1970's. The editor was originally called TVX, and the command set was modeled after the TECO editor. See, like Vi, has command mode and insert mode, and is normally in command mode. The commands are mnemonic. U for up, L for left, F for find, etc. It is very good for touch typists, and minimizes the need to move your fingers from the home row of the keyboard. If you don't have a favorite editor yet, or if you don't have a command mode editor you like, consider giving the See command set a try. It is really an efficient way to edit.

To use the See command set, start VIDE and select 'See' in the Options:Editor dialog. VIDE will remember your selection.

See is normally in Command mode. You can supply a count value to many commands by entering a value before the command. For example, entering 35d will move down 35 lines. When you enter insert mode, keys you type are inserted until you press the `ESC` key. The `f` find command lets you enter a find pattern that is echoed on the status line, and can include tabs. Press `ESC` to begin the search. The `F` version of find displays a dialog to enter the pattern.

In most cases, you can use a standard dedicated key (such as the arrow keys) as well as the equivalent mnemonic See command. You can highlight text with the mouse, and cut and paste in the usual fashion.

The See Command Set [top](#)

Key	Command Description
	<i>Movement Commands</i>
nl	Move left [Left Arrow]
nr	Move right [Right Arrow]
nu	Move up to beginning of line/TD>
nd	Move down to beginning of line
$n^{\wedge}U$	Move up [Up Arrow]
$n^{\wedge}D$	Move down [Down Arrow]

VIDE

<i>n</i> [Move left a word
<i>n</i> Tab	Move right a word [Ctrl–Right]
<i>n</i> ^P	Move a screenful up [PgUp]
<i>n</i> p	Move a screenful down [PgDn]
,	Goto beginning of line [Home]
.	Goto end of line [End]
b	Goto beginning of file [Ctrl–Home]
e	Goto end of file [Ctrl–End]
j	Jump back to previous location. This is an easy way to get back to where you were before a find command, or a large movement.
<i>n</i> ^L	Goto line <i>n</i>
m	Center Cursor in screen.
<i>mn</i>	note (mark) location <i>n</i> . Locations go from 1 to 25. This is a bookmark feature.
<i>n</i> ^N	Goto noted location <i>n</i>
<i>n</i> Ctrl–PgUp	Scroll a screenful up
<i>n</i> Ctrl–PgDn	Scroll a screenful down
	<i>Searching commands</i>
<i>n</i>]	Balance match. Find the matching
	paren, bracket, or brace.
f	Find pattern (non–dialog). This form of find allows you to enter the find pattern directly from the keyboard. The find pattern is terminated with the ESC key. The pattern is echoed on the status bar. This form of find is useful for Q–Macros.
F	Find pattern (dialog)
;	Find next
^F	Find/replace (dialog)
	<i>Insertion Commands</i>
<i>ni</i>	Insert char with value of <i>n</i> . This lets you enter Escapes or other non–printing characters.
i	Enter insert mode
Esc	Exit from Insert Mode
^O	Toggle insert/overtime [Ins]
<i>no</i>	Open a new blank line
	<i>Kill/change commands</i>

VIDE

^C	Copy highlight to clipboard
^V	Paste from clipboard
^X	Cut highlight to clipboard
'	Delete to line beginning
\"	Delete to line end
n^K	Kill line
n~	Toggle case
/	Kill 'last thing'. The 'last thing' is determined by the previous command. Commands that set the last thing include find, move a word, save, append, and get. Thus, a common way to perform a replace is to find a pattern, then use '=' to insert the replacement.
=	Change 'last thing'. Delete last thing and enter insert mode.
nt	Tidy (beautify) n lines. This will beautify C/C++, and Java code according to the V style conventions. To use this command, start on a code line that is indented how you want it. After that, 'n' lines will be formatted based on the starting indentation. This command is very useful if you use the V indent style. You can set the indentation for braces in the Options:Editor dialog. For text files (including HTML), this command will fill lines to the column specified in the Options:Editor dialog.
nBackspace	Delete previous character
nDel	Delete next character
	<i>Save buffer commands</i>
ns	Save n lines in save buffer. See provides a buffer that allows you to save lines. Using the save buffer is often easier than cut and paste. After saving lines, you can use the '/' delete last thing command to delete the lines you just saved. The See save buffer is independent of the clipboard, and is local for each file being edited.
na	Append n lines to save buffer
g	Get contents of save buffer. Inserts the contents of the save buffer at the current location.
y	Yank file to save buffer. This is an easy way to import text from an external file.
^y	Write save buffer to file. This lets you save part of your file in a new external file. Using 'y' and '^y' is often an easy to copy parts of one file to another, or include standard text into new files.
	<i>Macros</i>
\<a-z	Set register to use (a-z)
q	Record keystrokes in Q-Register until ^Q
n@	Execute current Q-Register N times
	<i>Misc. commands</i>

v	Repaint screen
?	Help
^Q	Save and close

Macros for See [top](#)

This section gives some specific examples of using Q-Macros with the See emulation.

First, consider search and replace. Even though VIDE has a search and replace function on the menu, you can do the same thing with a Q-register macro. Consider the following set of *See* commands:

```
fthe$=THE$
```

The '\$' represents the `ESC` key. You would enter this macro by selecting a Q-Register (e.g., `\a` to select Q-Register 'a'), entering a 'q' command, then entering the command sequence, and ending the recording the sequence with a Control-Q (`^Q`). The characters you enter will be echoed on the status bar (which will indicate you are recording). Then execute the macro (22 times, for example):

```
\a22@
```

The next 22 occurrences of 'the' will be changed to 'THE'. The corresponding command in Vi would be: `/the<CR>x!THE$` where '`<CR>`' is a Return, and '\$' is 'Escape'. Then execute it with `22@a`. Note that in this emulation of Vi, 'x' will delete the pattern just found.

If you want to execute the macro for the whole file, give a very large count. The macro will exit after any command, such as a find, fails.

This is a See example for placing a '***' at the beginning of every line that contains the word 'special'.

```
\aqlspecial$,i** $d^Q and execute \a1000@
```

The same function in Vi:

```
qa/special<CR>0i** $j0^Q and execute 1000@a
```

Future additions [top](#)

Things missing from the editor that WILL be included in future versions:

- Auto indent for C/C++ code
- Selectable highlight colors
- Formatted source code printing

The Vi Editor Emulation [top](#)

VIDE now includes an emulation of the Vi command set. The emulation is not yet complete, and it is likely it

VIDE

will never be a complete emulation of Vi. However, it is a pretty good "finger memory" emulation. For the most part, the right things will happen when you edit using your automatic "finger memory" of Vi commands. This emulation should improve over time. See the limitations section for a description of the current limitations and differences of this emulation.

To use the Vi emulation set, start VIDE and select 'Vi' in the Options->Editor dialog. VIDE will remember your selection.

The Vi Command Set

	(* after cmd means emulation not exact)
Key	Command Description
	*** Movement Commands ***
h,<Left>	cursor N chars to the left
j,<Down>	cursor N lines downward
k,<Up>	cursor N lines up
l,<Right>,<Space>	cursor N chars to the right
m<a-z>	set mark <a-z> at cursor position
CTRL-D	scroll Down N lines (default: half a screen)
CTRL-U	scroll N lines Upwards (default: half a screen)
CTRL-B,<PageUp>	scroll N screens Backwards
CTRL-F,<PageDown>	scroll N screens Forward
<CR>	cursor to the first CHAR N lines lower
0	cursor to the first char of the line
\$	cursor to the end of Nth next line
<Home>	line beginning
<CTRL-Home>	file beginning
<End>	line end
<CTRL-End>	file end
B*	cursor N WORDS backward ['word' not same]
"b	cursor N words backward
`<a-z>	cursor to the first CHAR on the line with mark <a-z>
``	cursor to the position before latest jump
'<a-z>	cursor to the first CHAR on the line with mark <a-z>
"	cursor to first CHAR of line where cursor was before latest jump
<MouseClicked>	move cursor to the mouse click position

VIDE

*** Searching commands ***	
/ {pattern} <CR>	search forward for {pattern}
/ <CR>	search forward for {pattern} of last search
? {pattern} <CR>	search backward for {pattern}
? <CR>	search backward for {pattern} of last search
N	repeat the latest '/' or '?' in opposite direction
n	repeat the latest '/' or '?'
%	go to matching paren (){}[]
*** Insertion Commands ***	
A	append text after the end of the line
a	append text after the cursor
i, <Insert>	insert text before the cursor (until Esc)
O	begin a new line above cursor and insert text
o	begin a new line below the cursor and insert text
R*	toggle replace mode: overwrite existing characters [just toggle]
CTRL-C	Copy to clipboard
CTRL-V	Paste from clipboard
CTRL-X	Cut to clipboard
*** Kill/change commands ***	
C	change from cursor position to end of line, and N-1 more
cc	delete N lines and start insert
c[bBhjklwW\$0]	delete Nmotion and start insert
D	delete chars under cursor until end of line and N-1 more
dd	delete N lines
d[bBhjklwW\$0]	delete Nmotion
J*	Join 2 lines [2 lines only]
S	delete N lines and start insert;
s	(substitute) delete N characters and start insert
u	undo changes
X, <BS>*	delete N characters before the cursor
x, 	delete N characters under and after cursor
~	switch case of N characters under cursor

*** Yank buffer commands ***	
P	put the text on line before cursor
p	put the text on line after the cursor
Y*	yank N lines; synonym for 'yy' [cursor at end]
y<	read file into yank buffer
y>	write yank buffer to file
*** Misc. commands ***	
CTRL-L	redraw screen
ZZ	save file and close window
gb	beautify N lines of C/C++/Java code, fill text
gm	center cursor in screen
"<a-z>	set q-register/buff for next op
["<a-z>]q	record to q-register until ^Q
["<a-z>]@	execute q-register N times

Emulation Limitations [top](#)

As much as possible, this emulation tries to do the same thing Vi would do with the same command. Probably one of the main things missing in this emulation is the total lack of **Ex** support, i.e., there are no ":" commands supported. In fact, this should be a minor limitation as there are usually menu commands that support the most important ":" equivalents. You can also record [macros](#) that can duplicate many of the functions typically done with ":" commands.

Another major difference is the lack of support for a count for insert and find operations. Thus, "5ifoo\$" will not insert 5 instances of "foo". This is true for some other commands as well. Generally, if you enter a command expecting some kind of repeated operation, and the VIDE Vi emulation does not support repeat, then the operation will be done only once. Again, you can use macros to get around this limitation.

The following list summarizes some of the other differences in how this emulation works.

- **c** and **d** commands are somewhat limited. The currently supported motions after the 'c' or 'd' include: 'b', 'B', 'h', 'j', 'k', 'l', 'w', 'W', '\$', '0', as well as the 'cc' and 'dd' forms. *Currently, only the full line motions add text to the yank buffer.* Most of the other somewhat obscure motion commands used with 'c' and 'd' can often be done using the mouse to cut and paste.
- The **y** command currently supports only the 'yy' form of the command. In addition, the new VIDE version commands 'y<' and 'y>' have been added. These are used to support external files. Use 'y<' to read an external file into the yank buffer, and 'y>' to save the contents of the yank buffer to an external file.
- The **Backspace** key deletes the character in front of the cursor like almost all other apps you are likely to use instead of moving left. A little note: The backspace key is really CTRL-H. For consistency, the original Vi used CTRL-H and 'h' as left cursor. Now why are 'h', 'j', 'k', and 'l' used for cursor movements? Because the computer lab at UCB where Vi was written had mostly ADM-3a terminals.

VIDE

(ADM-3a terminals were far cheaper than most other terminals available at the time.) And the 'H', 'J', 'K', and 'L' keys had cursor arrows printed on each of them. Thus, the "hjkl" commands for cursor motion.

- Cut, Copy, and Paste are implemented using the standard system clipboard. The standard 'CTRL-X', 'CTRL-C', and 'CTRL-V' commands are used to support clipboard operations.
- The find commands will highlight the pattern found. If you press 'X' or 'x' while the pattern is still highlighted, the pattern will be deleted. While this is different than standard Vi, it is really handy for search and replace.
- This emulation doesn't use the visual '\$' marker for some kinds of edits on partial lines, it just does the edits. Another left over from the ADM-3a days, I think.
- Left and right cursor movements flow to the adjacent lines. The V editor class just doesn't support movement limited to one line.
- The 'J' command will only join 2 lines.
- The find command (/) echoes the pattern on the status bar, and not in the text window. You can use an 'Esc' to terminate the pattern as well as a 'CR'. Find does not support regular expressions. Selecting find from the tool bar or menu brings up a dialog box instead of using the status bar.
- Use menu command or the tool bar button for find/replace.
- The 'gt' command formats C, C++, and Java code. See a more complete description of the See 't' command. Right now, the formatting is limited to the V standard, but support for other styles will likely be added.
- The 'R' command toggles Insert and Overtyping mode. It doesn't force overtype (replace) mode.
- The '%' command works a bit differently. It currently only supports paren-like characters, and you must place the cursor on the paren you want to match.
- The cursor doesn't change shape for Insert and Normal modes. The mode is shown on the status bar.
- You need to use the menus to load and save files.
- VIDE's concept of a 'word' is different than Vi's. There is no difference between 'w' and 'W', for example. This might get fixed. Because you get visual feedback, this probably isn't a big problem.
- The Vi emulation was built starting with the See command interpreter code. This saved lots and lots of work with minimal compatibility issues. One however, involves counts. The current code can't distinguish from the default count of 1 and an explicitly entered count of 1. Thus, commands that use default counts might not work correctly when explicitly given a count of 1. For example, the 'G' command can't go to line 1 -- it goes to the last line.
- The 'gm' command is used to center text on the screen.

The following list summarizes most of the Vi features that have not yet been implemented. They are likely to be added in future releases.

- The '.' command is not supported. I hope I can figure out a way to add it, but the basic V editing class is not very compatible with this command.
- There is only one yank buffer. Named buffers are not yet supported.
- If there are Vi features that are important to you, please let me know by e-mail, and I will try to add them. I won't add '.' command support, but I may add the equivalent support via an extended command set. If some of the emulated commands don't quite work how you expect, let me know and I will try to make them work more closely to Vi.
- If you really want to add some features (like '.' support), the source code is available! Fix away!

No Warranty [top](#)

VIDE

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler
All rights reserved.

Bruce E. Wampler, Ph.D.
bruce@objectcentral.com
www.objectcentral.com

VIDE C/C++ Tutorial

- [VIDE Overview](#)
 - [Using VIDE with GNU gcc/g++](#)
 - [Using VIDE with Borland C++ 5.5](#)
 - [VIDE Projects](#)
 - [VIDE Help System](#)
 - [Debugging with VIDE](#)
 - [No Warranty](#)
-

VIDE Overview

The design of VIDE has been somewhat evolutionary, but you should find that it is not that much different than other IDEs you may have used. Because VIDE is a free, open source program, it probably lacks some of the polish of commercial IDEs. However, it is still quite functional, and it is really easier to develop programs with it than it is to use a command line interface.

Generally, any application you write will consist of various source files (with associated header files for C/C++), and required data files. These files are generally dependent on each other. By defining a VIDE Project for your application, all the file dependencies are automatically handled. The standard tool `make` is used for C/C++ files, while the JDK Java compiler automatically handles dependencies.

Using VIDE, the normal work cycle goes:

1. Design your application.
VIDE currently has no capabilities to help with this stage.
2. Start VIDE, and create a Project File.
This will include all source files, compiler options, and other information needed to compile your application.
3. Build your project.
This stage compiles your source into object code. Compilation errors are displayed in the status window, and you can simply right-click on the error to go to the offending line in your source code. After making corrections, you repeat this step until all compilation and linking errors are removed.
4. Run your program.
You can start your program from within VIDE.
5. Debug your program.
VIDE for has integrated support for the **`gdb`** debugger for C/C++ on both MS-Windows and Linux. On Linux, you can specify `DDD` as your preferred debugger in the Options menu. VIDE also supports **`jdb`**.
6. Write documentation for your application.
VIDE has syntax highlighting for HTML to make that job easier. You can also automatically launch your web browser to view the resulting HTML pages. Really neat.

Using VIDE with GNU gcc/g++ [top](#)

The main C/C++ compiler VIDE is designed to work with is the GNU Compiler Collection (`gcc`), either on a Unix-like system, or on MS-Windows with the GNU GCC compiler. (VIDE will work with either the

MinGW version of GCC, or the Cygnus version. Try the latest merged GCC 2.95.2!) Functionality of VIDE for gcc is based on standard GNU makefiles. VIDE uses a standard GNU make Makefile to build your project. Thus, you must have a Makefile defined. This Makefile can be one created automatically by VIDE itself from your Project file, one generated by the vgen V application generator, or even one you've written yourself. If you have your own Makefile, then you probably won't need to use a VIDE Project.

VIDE assumes you have your gcc/g++ compiler already installed on your system and the PATH correctly set. For Unix/Linux systems, this is a given. If you are using a MS–Windows version (MingGW or Cygnus), then you must follow the instructions provided with their distributions. You might find it helpful to copy the VIDE executables to the /bin directory that has your g++ compiler. VIDE requires GNU make (it is not compatible with some other versions of make), the GNU C/C++ compiler (preferably the latest GCC 2.95) and associated utilities, and the GNU gdb debugger. You also may want the V utilities vgen and viconed.

Using VIDE with Borland C++ 5.5 [top](#)

Beginning with version 1.07, VIDE has added support for the free Borland compiler. This support includes support for running Borland's Turbo Debugger. More complete information about using VIDE with the Borland compiler is found in the [VIDE Borland](#) reference guide. It is *essential* that you read the information in that reference to get a properly working version of VIDE with Borland BCC32.

Functionality of VIDE for BCC32 is based on the non–standard Borland MAKE program. VIDE uses a Borland make Makefile to build your project. Thus, you must have a Makefile defined. This Makefile can be one created automatically by VIDE itself from your Project file, or one you've written yourself. If you have your own Makefile, then you probably won't need to use a VIDE Project.

VIDE assumes you have your Borland compiler already installed on your system and the PATH correctly set. When working with the Borland compiler, VIDE requires the Borland make.

VIDE Projects [top](#)

General procedures

Once you start VIDE, you will see a blank window labelled "No Makefile, Project, or .java file Specified." This opening window is the message window, and is used to output the results of your make. A typical first step after starting VIDE is to open a VIDE Project or select an existing Makefile.

Once you've opened a project or specified a Makefile, you can build your project with the **Build:Make C++** menu command, or click on the Make button on the tool bar. This runs the make utility with the default target (often "all"). VIDE first runs the Makefile in dry run mode. It uses that output to then run the commands generated. It intercepts the error messages for g++ and put them in the message window. You can then right–click the error line, and VIDE will open up the file in question, and put the cursor on the offending line. This all assumes that the source and makefile are in the same directory. VIDE also assumes all your files have unique names (i.e., you don't have files in your project with the same name but in different directories). After you correct the problem, rerun make.

You can also make a specific target in your makefile by using the Make menu: **Make:Make <target>**. If you include a "clean" target in your Makefile, **Make:Make Clean** will run make clean. (The Makefile generated by VIDE from a Project has a "clean" target.)

The tools menu allows you to run your program. If you are using a VIDE Project, then **Tools:Run project** will run the project you just built. If you are using your own Makefile, then **Tools:Run project** will prompt you for the file to run.

VIDE C/C++ Projects [top](#)



When you are first creating a new project (or moving an existing program to VIDE), click on the **Project:New C++ Project** menu. After you select a name for your project, a "wizard" dialog will open that will let you begin defining a project. The first thing to fill in is the name of the program you are building (*Target name*). If you are building an application, this will be something like "foo.exe" on Windows, or simply "foo" on Linux or Unix systems. Libraries end with a ".a" extension: "foo.a" on both platforms, or a ".lib" for Borland.

Next, select the type of project you want to build. A *Console Application* runs in a shell or console window, and does not use any GUI components. A *GUI application* uses a graphical interface. On Windows, this will be the standard WIN32 API, while on X systems, you will have a choice of Athena, Motif, or gtk. You can also use the V GUI. You will be given a choice between using the static V library or a dynamically loaded library. If you are using OpenGL or Mesa, check that box.

You can also build a static library. Static libraries are usually easy to build. Both Windows and Linux support dynamic libraries (called DLLs on Windows and shared libraries on Linux). However, the rules for defining and getting a shared library (especially DLLs) is somewhat beyond the scope of VIDE. The VIDE project file can support DLLs and shared libraries, but you have to just how to put a dynamic library together first. See the later section on Advanced options for a few more details of building a DLL.

You can also select which you are building, a release version, or simply a debugging version of your project. These options simply determine some switches to the compiler. VIDE does not support both a release and a debug build within the same session. The easiest work around for this is to first build one version, then use **File->Save As** to save the alternate version.

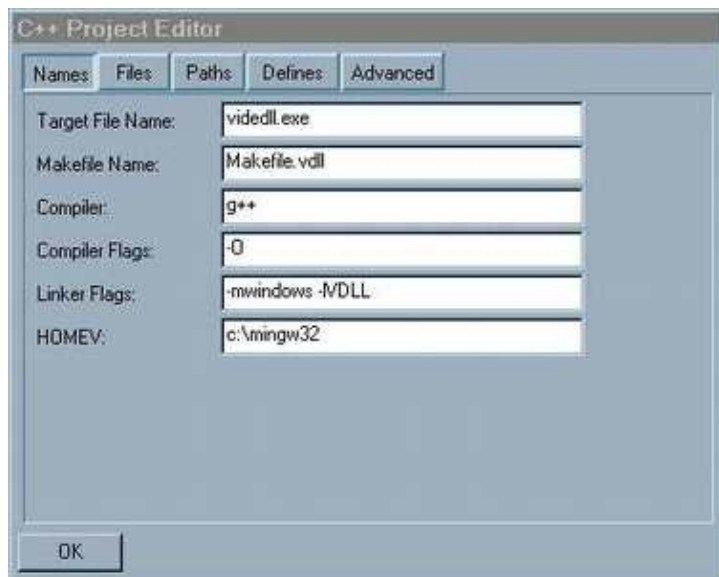
VIDE

The Windows version also lets you select which compiler you are using – MingGW, Cygnus, or Borland. Setting the "-mno-cygwin" option will build an application under Cygwin without using the Cygwin DLL.

Once the initial project attributes have been selected, You will get the project editor dialog box with various tabbed items. Most of the fields will be filled in according to the values you set in the opening dialog. The main thing you will probably want to do is add source files using the Files tab. You can set defines as needed in the other tabs. Once you have added the files needed and click "Done," VIDE will create a Makefile suitable to compile your project with gcc or Borland BCC32.

Each of the project editor tabs are described in more detail in the following sections. (These screen shots were taken from the project file that builds VIDE itself using the V GUI library.)

Names



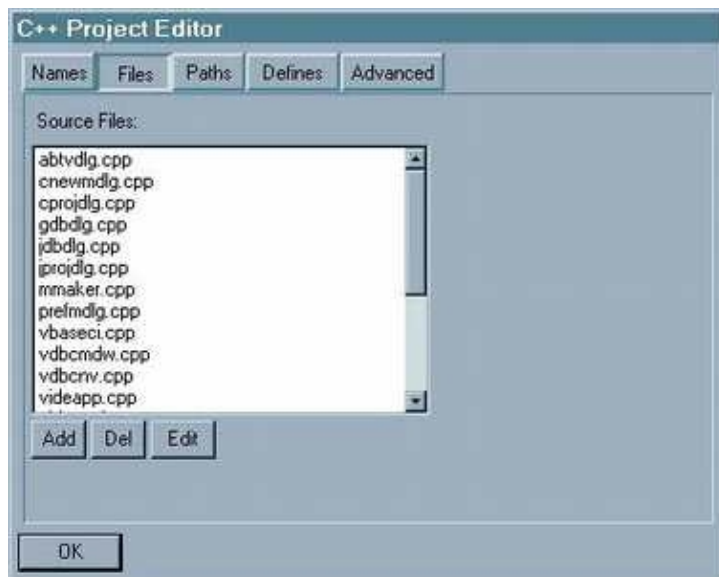
This pane lets you set the target name for the executable or library. You can also change the name of the generated Makefile. Usually, you will use g++ as the compiler. The Borland compiler is BCC32. The target name determines what kind of final target is built. A ".exe" extension on Windows, or no extension on Linux, causes an executable to be built. If the target name ends in ".a" (or ".lib" for Borland), then a static library will be built. It is important to get these extensions right to generate the correct kind of target.

Compiler Flags line lets you pass switches to the compiler, such as -O for optimize, or whatever. The linker flags are passed to the linker, and usually consist of a set of library references. The new project wizard will usually fill these fields in as needed for console, GUI, and V apps. The HOMEV value is required for programs that use the V library if the V GUI system has not been installed in the same places as other libraries and include files on your system.

You may have to change some of the default switches for your specific compiler or operating system. You should only have to do this once for a project. If you will be creating other projects, you can save a template project in a file of your choice, and then use it as a starting point for new projects.

The [Borland guide](#) gives some specific details for the Borland version.

Files



This lets you add the names of the source files included in the project. Clicking ADD brings up a file selection dialog. When you select a file, the file is added without any path name. To delete the selected entry, use the Del button. Until V adds multi-line selection (someday soon), you have to add files one at a time.

On Windows, the source file can be a ".rc" resource file as well. If you include a ".rc" file, VIDE will automatically add the dependency to the Makefile, and use the appropriate resource compiler (WINDRES for gcc or BRCC32 for Borland) to compile a ".o" file for gcc or a ".res" file for Borland.

Note: Adding relative file paths

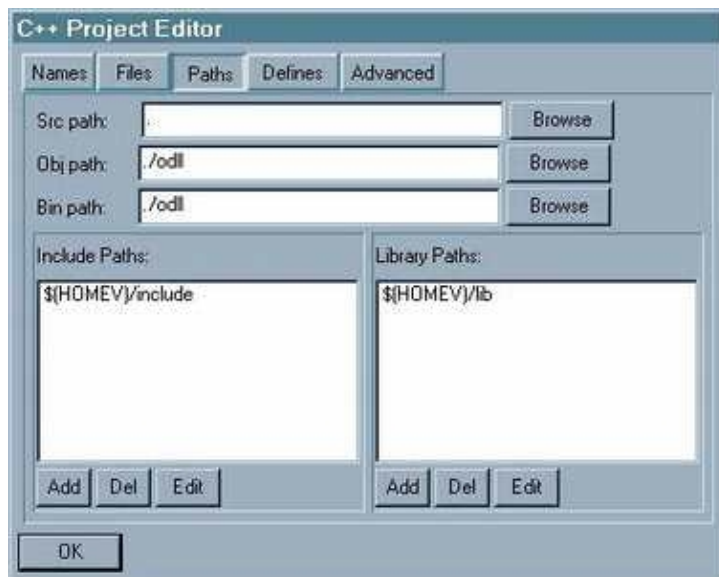
The current mechanism for adding files does not support adding relative or absolute paths in front of a file name using the file selection dialog. Normally, VIDE assumes that you will have your source files in the directory specified in the source directory path. Thus it strips any leading path. However, you *can* add relative paths. If you need to add a relative (or absolute) path, first add the file using the file selection dialog. Then, select the file from the list file, and click Edit to hand edit the entry. Then add the relative path to the file.

Note: Multiple source directories

Sometimes you may want to keep files in multiple directories. With GNU make, this is not a problem. Simply use relative paths. Borland Make has some problems, and doesn't seem to handle embedded relative paths. So, you can specify an exact path if you wish. Simply edit the file, give the full path, but put a leading = (equal sign) in front of the file name. When generating the Makefile, VIDE will then copy the file name literally, and not append the standard "\$ (Src)" symbol.

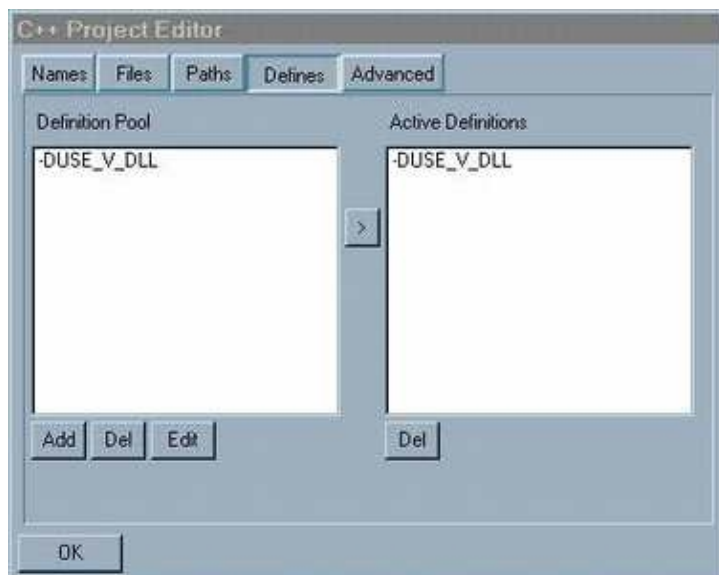
(Note: using the file selection dialog can leave the current directory set to something other than the default source directory, and when you exit the project editor, the generation of the Makefile may fail because things get started in the wrong directory. This problem is a bit hard to fix, and only happens when you use the file dialog box to select a file from a different directory. Just edit the project again, and the Makefile will be generated correctly. Someday I may fix this, but I have higher priorities for now. Sorry.)

Paths



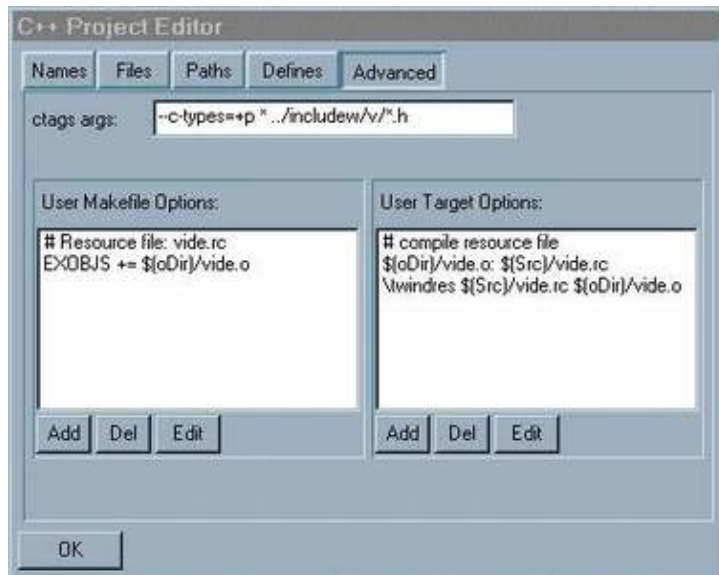
This lets you specify the directory for the source files, the directory where you want object files to be generated, and the directory where the binary should be written to. It is best to use relative paths for these whenever possible. You can also list paths for include and library directories. These are passed to g++ as the appropriate switches.

Defines



It is often helpful to provide compile time defines for C and C++ programs. This tab lets you add compile time defines. The left list shows a pool of definitions that you might want to use. To have them included at compile time add them to the Active Definitions list using the ">" button between the lists. The pool will stay constant, while the Active side can be a bit more dynamic as you work with your project. Note that you have to supply the full g++ definition switch: `-DFOO` or `-UNOTFOO`.

Advanced



The first field of the Advanced pane lets you define a project specific argument for the ctags program. For example, the entry `"-c-types=+p * ../includew/v/*.h"` does several things. First, it adds extra information about function prototypes. Then it includes the V library headers in the tags file for easy lookup of functions used from the library. Check the ctags documentation for other switches you might want to use.

The second advanced field lets you specify startup switches for the debugger. (Not shown in this screen capture.) For example, when used with gdb, `"-nw"` is provided by default to start gdb in non-windows mode.

The other advanced panels let you add lines to the Makefile in one of two places. You will have to have a pretty good understanding of Makefile in order to make effective use of these options.

Anything you add to the "User Makefile Options" list will be written to the generated Makefile immediately after the standard definitions. You could use it to define your own symbols, or whatever.

The "User Targets" list lets you add new targets other than the defaults to make. You can use these to define specific values you want to add to your Makefile. The Borland support uses these to define the default runtime libraries.

If you want to build something other than an executable or a library, there is one important feature provided by the user targets list. If the *first* entry has `"#all"` in it, then VIDE will not generate an "all" target (usually the same name as the target name) for the make file. It assumes you are providing the all target here instead.

Note that when you need a leading tab for the makefile, enter a `'\t'` into the project. It will be automatically converted to a real tab in the final makefile.

Direct Project File Editing

A VIDE `.vpj` project files is in fact regular text file. It is laid out in clearly labeled sections. While you can add entries to any section using the Project Editor, advanced users may find it easier to edit the V project file directly to add definitions and targets that aren't generated automatically. Using these two mechanisms (defines and targets), you can build complicated makefiles which will be automatically generated from the

project file. It is possible to define practically anything you might need to include in a makefile. For example, all the options needed to generate the Windows DLL version of V are included in a VIDE project file.

I'm not going to explain every detail of the project file format here. Any programmer reasonably familiar with makefiles will be able to see some of the potential of the VIDE project file. For example, VIDE uses a set of standard variable names such as "oDir" and "EXOBS". The standard make targets are defined using symbols, too. It is easy to define entries in the user defined symbol section that use the makefile "+=" operator to modify and add to the standard symbols. You can add very complicated targets to the user target section, especially when you override the "all" target with the "#all" convention. Perhaps the best thing to do is look at a VIDE Project File, and study the VIDE and VGUI project files for examples of some of the things you can do. The main advantage of using a project file is the automatic generation of dependencies and other features that will be eventually included in VIDE.

VIDE Help System [top](#)

VIDE now includes a Help menu. Most of the help is supplied in a separate distribution file, and is in HTML format. V Help uses your default Web Browser to show the help files.

I have attempted to collect the most useful documentation I could find for the various GNU C++ tools. If you download and install the Help files, you should have a very complete and useful set of documents for C, C++, and Java programming at your finger tips. See [Installing VIDE](#) for more instructions on how to install VIDE help.

If there are other HTML based documents you would like added to the VIDE distribution, please let me know.

Debugging with VIDE [top](#)

VIDE supports the standard GNU **gdb** and Sun **jdb** debuggers. The VIDE interface to the debuggers makes it far easier to debug your code, but is of minimalist design. The goal is to make using the native debuggers as easy as possible for casual users, while maintaining the full power of the debugger for experienced users. VIDE accomplishes this by showing a command window interface to the debugger. You can enter any native debugger command in this window, and thus have full access to all debugger features.

VIDE makes using the debugger easier by providing a popup dialog with the most often used commands. And most importantly, VIDE will open the source file in an editor window and highlight the current execution line on breakpoints or steps. It is very easy to trace program execution by setting breakpoints, and clicking on the *Step over* or *Step into* dialog buttons. VIDE also allows you to inspect variable values by highlighting the variable in the source and right clicking the mouse.

A description of debug dialog commands is provided in the [VIDE Command Reference](#) section.

Debugging C/C++ with gdb

To debug C/C++ programs with **gdb**, you must first compile the program with debugging information. This is accomplished with the **-g** switch on the compile line. The current version of VIDE does not provide automatic generation of debug or release makefiles. The easiest way to define VIDE projects for both debug and release versions is to use the **Project:Save Project as...** command. First, define a release version of the project. Then, using that project as a template, change the switches as needed for your debug version, and save

the project under a different name.

The full power of gdb is available in the debugger command window. You may enter any standard gdb command after the "(gdb)" prompt. In fact, there really is limited interaction between VIDE and gdb, mostly handling breakpoints. VIDE starts gdb using the "-f" switch, which causes gdb to send a special output sequence after each break, which VIDE then uses to open and display the highlighted break line.

VIDE maintains its own list of breakpoints, which it keeps even if you start and stop the debugger. It is important that you use VIDE commands to set and delete breakpoints. If you enter breakpoints directly into the gdb command window, VIDE won't know about them, and won't highlight them in your source code.

MS-Windows gdb 5.0

At the present time (June, 2000), gdb Version 5.0 has been released for Windows. At this time, the only ready to run version is supplied with the Cygwin gcc distribution. Since gdb 4.18 supplied with MinGW still seems to have problems debugging large Win apps, I've made a stripped down version of the Cygwin gdb version available on the ObjectCentral web site. There are actually two versions. One is the full Cygwin version with the tk/tcl windowed interface, and the other only works in console mode (and must be started with the -nw switch). The stripped down version works great with VIDE for both Cygwin and MinGW code. In fact, I can finally actually debug VIDE itself using this version of gdb!

Limitations with gdb

- [MS-Windows 9x/NT] Output from console applications isn't properly displayed in the gdb command window. If you run gdb from a command window, the output is displayed as it is generated. With the VIDE gdb command window, the output from the running program is not displayed until the very end when the program terminates. I have no idea why gdb behaves like this, but it must have something to do with the way I use CreateProcess. Any help on this problem would be appreciated.
- [MS-Windows NT] On Windows NT, gdb seems to bring up a message dialog trying to open some file over the network. I don't have a networked NT machine, so I don't know if this is a general gdb feature or what, but gdb seems to work if you just press the appropriate button to ignore the problem when the dialog is displayed.

No Warranty [top](#)

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler
All rights reserved.

Bruce E. Wampler, Ph.D.
bruce@objectcentral.com
www.objectcentral.com

VIDE



VIDE User Guide	Editor Reference	VIDE Java Tutorial
Command Reference	VIDE Version 1.23 - 24 Jul 01	VIDE C++ Tutorial

Using VIDE with the Sun JDK

- [VIDE Overview](#)
 - [The Development Cycle](#)
 - [Java and the Sun JDK](#)
 - The Development Cycle using VIDE
 - ◆ [Defining the VIDE Java Project](#)
 - ◆ [Compiling your program](#)
 - ◆ [Running your program](#)
 - ◆ [Debugging with VIDE](#)
 - [VIDE Help System](#)
 - [No Warranty](#)
-

VIDE Overview [top](#)

An Integrated Development Environment (IDE) is a software tool intended to make the process of writing programs easier. An IDE typically works as a unit with your compiler environment. It allows you to edit your program source code, compile it with the compiler, fix syntax errors, run it, and debug it, all from inside a consistent and convenient environment.

VIDE is somewhat different than other Java IDEs such as Borland's JBuilder or Sun's Forte. VIDE is much smaller than those IDEs, and simpler to use. VIDE is not really intended for giant Java projects (although it still could be used for a big project.) Instead, VIDE is meant to be a step up from using the command line and a simple editor, and a full blown IDE. It can make things simpler.

VIDE has been designed to work with Sun's JDK (Java Development Kit). Instead of using the compiler and interpreter from command shells (MS-DOS prompts), you interact with them using VIDE. The entire development process is simplified.

I will discuss the specifics of Sun JDK later (if you have programmed before, or if you have used an IDE before, you can probably skip to that section right now). But first, here's the general procedure for building a working Java application using VIDE and Sun's JDK.

Generally, any application you write will consist of various source files, and required data files. In order to get your program to compile and run, you must have all the files available. To track all the files, and to set various options required by the compiler, you will create a VIDE Project for your application. The VIDE Project file contains all the information needed to build and run your application using the JDK tools.

Thus, using VIDE, the normal work cycle goes something like this:

- 1. Design your application.**

Sorry, you have to do this part yourself. While there are software tools that can help you do this, VIDE currently has no capabilities to help with this stage.

- 2. Start VIDE, and create a Project.**

You should almost always start with a new Project, even if you don't yet have any source code entered. It is best to name the project the same name as your Java application name. By creating a

new project, you will define the application name, and thus the name of the main source file. You also can define compiler options, and other information needed to compile your application.

3. Create your source code.

Typically you will need to create the source code. VIDE provides an excellent programmer's editor. Programmer's editors are really somewhat different than a generic text-pad like editor, or even a word processor. Typically, they have commands appropriate for programming, and support special features like syntax highlighting and automatic program formatting. VIDE is no exception. It has syntax highlighting, program formatting, and an extensive help system designed just for programmers.

4. Build your project.

Once you have your programs entered, you will need to compile your source to object code. It is almost certain that your code will have both typos and logic errors. When you compile your code from within VIDE, you will see compilation errors displayed in the status window. You can simply right-click on the error, and VIDE will open the source code file, and go to the offending line. After making corrections, you repeat this step until all compilation and linking errors are removed.

5. Run your program.

You can start your program from within VIDE by clicking the run icon.

6. Debug your program.

Your program will almost certainly have logic errors in it as well. A debugger is used to help you find the logic errors in your program.

7. Write documentation for your application.

Once you have an application running and tested, you will likely need to write documentation for it. VIDE provides some extra support for HTML, including syntax highlighting. You can also automatically launch your web browser to view the resulting HTML pages. Really neat.

Java and the Sun JDK [top](#)

Java Basics

Getting a Java program to run on your machine normally takes several steps. First you create the source file (using the VIDE editor). That source code is then compiled by the Java compiler `javac`, which produces Java bytecode output. Java bytecode can be run on any computer that supports Java. Using the JDK, you typically execute your bytecode by running the Java bytecode interpreter. There are *two* versions for MS-Windows: `java` (for console apps), and `javaw` (for GUI window apps). On Linux, because applications are started much differently than on MS-Windows, there is just one: `java`.

If you have an applet, you need an HTML file to go with the applet bytecode, and then run the applet either by running your usual web browser, or using the JDK's `appletviewer`.

Typically, you run the compiler and interpreter from a command window. This is usually an MS-DOS `PROMPT` on Windows, or the normal command shell or console window on Linux. (Note that using VIDE, however, eliminates the need to explicitly use a command window.)

To summarize:

- Java console application:

Source code -> javac -> java

- Java MS–Windows GUI window application:

Source code -> javac -> javaw

- Java Linux GUI window application:

Source code -> javac -> java

- Java applet:

Source code -> javac -> + HTML -> browser

–or–

Source code -> javac -> + HTML -> appletviewer

Important Note: MS–Windows vs. Linux consoles

Linux and MS–Windows start programs in a fundamentally different way. Because of this, Sun supplies both `java` and `javaw` for MS–Windows. And because Sun supported the JDK for MS–Windows first, VIDE was designed to work best for that environment. This means there is a little difference between the Linux and the MS–Windows version.

When you run a Java console app from inside VIDE, you need to create a console shell to see your output. VIDE uses the fact that you've specified `java` as the interpreter to decide to run your app in a console. If you've specified `javaw`, then VIDE will launch your app so that it does not have a console. Even though the Linux version of the Sun JDK does not have a `javaw` interpreter, you still need to specify `javaw` for Linux GUI apps. When you create a new Java project, VIDE will automatically fill in `javaw`. When you actually run your app, VIDE will run the correct `java`, even though the project file says `javaw`. This slightly strange convention allows VIDE project files to be identical on MS–Windows or Linux.

Sun JDK Specifics [top](#)

As distributed, VIDE has been designed to use the default conventions of Sun's JDK 1.3 (Java 2). It should be easy to change the default values to whatever is required by earlier JDKs. VIDE assumes you have correctly installed JDK according to Sun's instructions, and have appropriately set whatever environment variables you need in your `AUTOEXEC.BAT` file on Windows, or the appropriate shell initialization file on Linux.

VIDE allows you to interact with the JDK tools from within VIDE rather than running all the JDK components directly from a command window. VIDE also takes advantage of some of the advanced features found in the JDK component. VIDE lets the JDK compiler `javac` handle all the file dependencies for projects that require more than one source file.

Once you start VIDE, the first action normally is to open a VIDE Java Project or select a Java source file. The

opening window is the message window, and is used to output the results of your compiles. If you have a small Java application (not an applet), you can simply specify the name of the top level Java class file from the **Project:Select Makefile or Java file** menu. However, it is almost always better to use a VIDE Java Project. If you have an applet or need to specify compiler options, you must create a new Java Project file.

Once you've opened a project or specified a Java source file, you can compile your project with the **Build:Compile Java** menu command, or click on the Make/Compile button on the tool bar. This runs the Java compiler, `javac`. The results of the compile are shown in the message window. If you get an error, you can usually right click on the error line and the source file will be loaded into an edit window, and the cursor placed on the offending line.

The Development Cycle using VIDE [top](#)

VIDE takes advantage of the way the Java compiler (`javac`) works. If you use `javac` from the command line, you can always compile every file in your program simply by entering "`javac YourProgram.java`". The compiler will compile your program, and then produce all the class files you need.

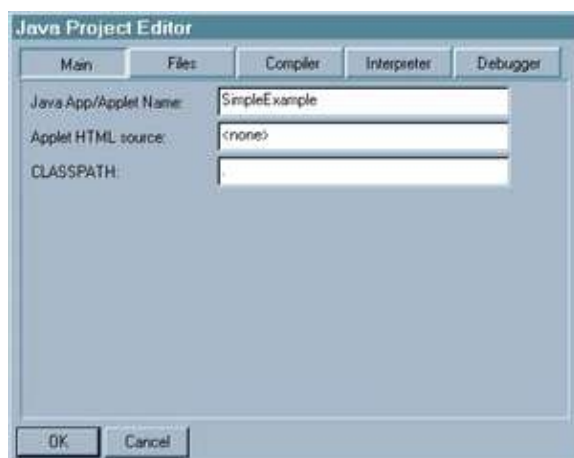
To keep things as simple as possible, VIDE works with only a single Java file – the one with your "main" method in it. It will usually be the easiest to call the project file by the same name as well.

Defining the VIDE Java Project [top](#)

When you are first creating a new project (or moving an existing program to VIDE), click on the **Project:New Java Project** menu. You will get a dialog box with various tabbed items. You will only need to use the *Main* and *Files* tabs for most projects. Once you've specified the required information, you can compile your project as described earlier.

Note that the name you specify on the "Main" tab will be the name of top level file in your program. You won't have to specify any other file names like you do in some other IDEs. The Java compiler knows how to find all the rest of the files you need.

Main



Java Project Editor: Main

This pane lets you specify the name of your application or applet. This should be the Java Class name of the

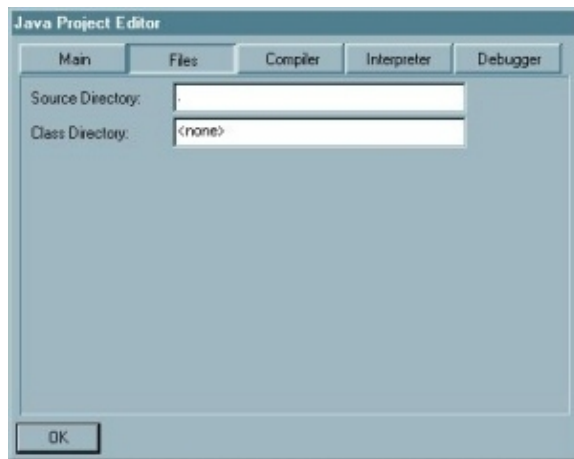
VIDE

top class. You don't need to add `.java` or other extension. If you are creating an applet, then you will need to specify the name of the file with the HTML code that runs your applet. When you run an applet, VIDE uses JDK's `appletviewer`.

If you have JDK 1.3, you usually won't need to use the `CLASSPATH` setting. However, earlier JDKs and some circumstances may require the `CLASSPATH` to be specified. This information is passed to the compiler and interpreter via switches. See Sun's documentation for more information about `CLASSPATH`.

HINT: When you open a new project file, VIDE assumes some options and switch settings that are commonly appropriate to use with the compiler. It is likely that the defaults will not be the one *you* want. There is an easy solution. Simply create a new project, change the settings to be just what you want, and then save that project as a "template" using an appropriate name. Next time, open that template project file, and immediately use **Project:Save Project As...** to save it under the working name of the new project.

Files



This pane lets you specify the source directory for your Java files, as well as the output directory for the generated `.class` files. Typically you will want to put all your source files and the VIDE project file in the same directory.

You can also leave the default values for the output directory, and the `.class` files will be generated to the same place as your source files. If you specify a directory, VIDE will pass it to the compiler using the `-d` switch.

Simply by putting your main Class definition in the top level source directory, you end up with everything compiled that needs to be. You can also put files in a subdirectory of the project file, and put the class files in another directory. Putting the source code for different packages in directories below the top level source also will work just how you want. But the very simplest thing to do is to leave the default values for these, and just put all your source into a single directory.

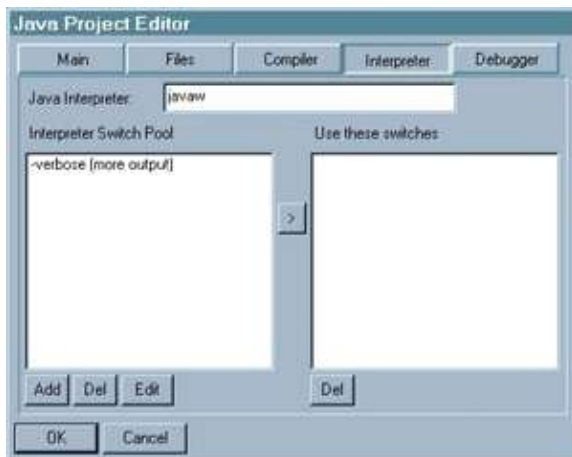
Compiler



Java Project Editor: Compiler

This pane lets you specify which Java compiler to use. It is almost always `javac`, but you can change it for other development environments. The Compiler Switch Pool has the standard switches supported by JDK 1.3. Click the `>` button to use a switch. You can add your own switches to the Pool if you need to (for other JDKs, for example).

Interpreter



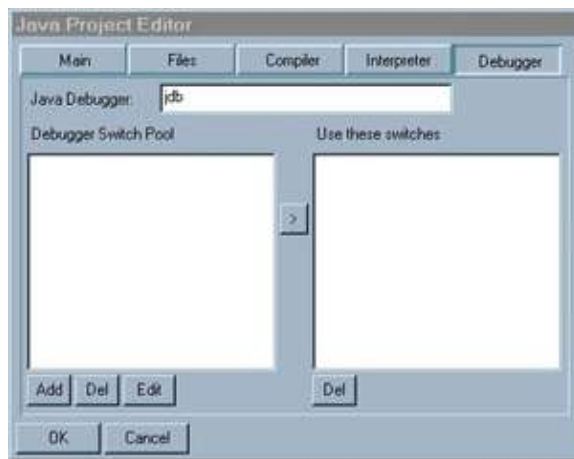
Java Project Editor: Interpreter

This pane lets you specify the interpreter to use to run your app. The default value is `java`, which is suitable for running console-type apps. If you use `awt` or `swing` for GUI base apps, you should use `javaw`. You can still use `java` to run GUI apps, but you will get an extra command window on the screen as well as your GUI window.

Note that you specify `java` or `javaw` on Linux, even though the Linux JDK does not have a `javaw`. VIDE will start your app correctly automatically: `java` means a console app, `javaw` means a GUI app.

Java applets are handled differently. To view an applet, you must specify the HTML source in the Main pane, which will cause VIDE to launch `appletviewer`. If you want to see your applet inside a browser window, you can edit the associated HTML file, and use **File:Send to Browser** to start your browser.

Debugger



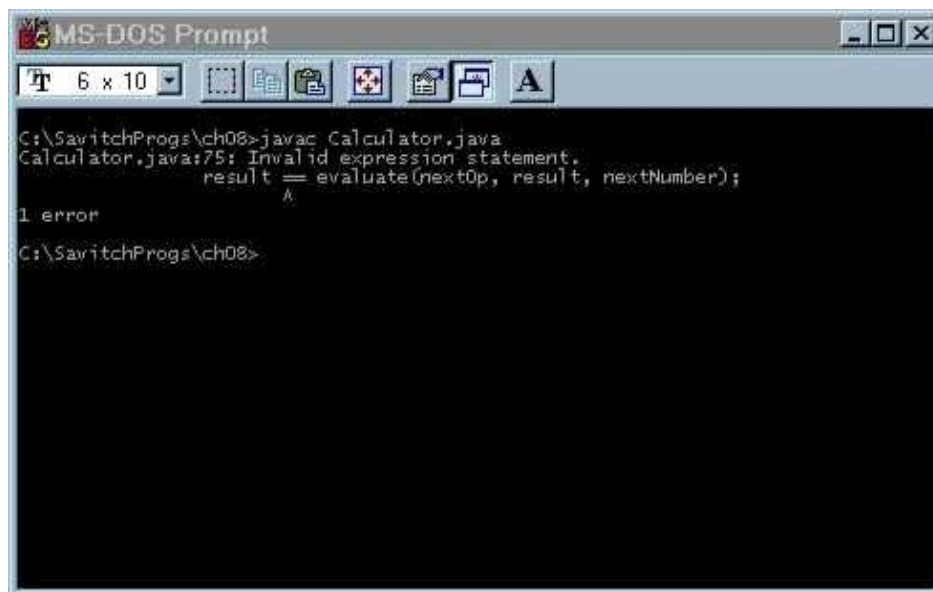
Java Project Editor: Debugger

This pane will let you specify options you may need for the debugger.

Compiling your program [top](#)

Once you have created your source files and created your project, you will compile your Java code from inside VIDE. It is this part of the development cycle that VIDE really shines and makes your life easier. All programmers, but especially beginning programmers, make errors when entering the source program. These errors result in what are called *syntax errors*, and cause the compiler to generate error output.


When you use the compiler from a command window, you first must enter the compiler command and the name of the file. For example, `javac Calculator.java`. The output of the compiler is then displayed in the command window as shown in the following figure:

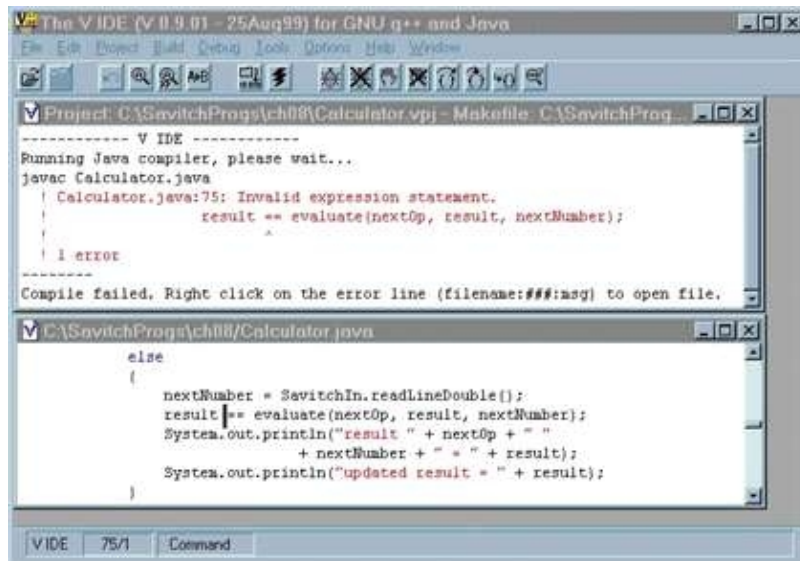


Java Syntax Error in Command Window

VIDE

Often, a single syntax error will cause the compiler to generate multiple errors. If there are too many errors, the error output often scrolls off the command window. And then you have to edit the source code by finding each line with an error.

Using VIDE is much simpler. To compile, you either use the **Build:MakeC++/Compile Java** menu command, or click on the build button . When you run the compiler from within VIDE, the resulting error output is put in the VIDE status window:

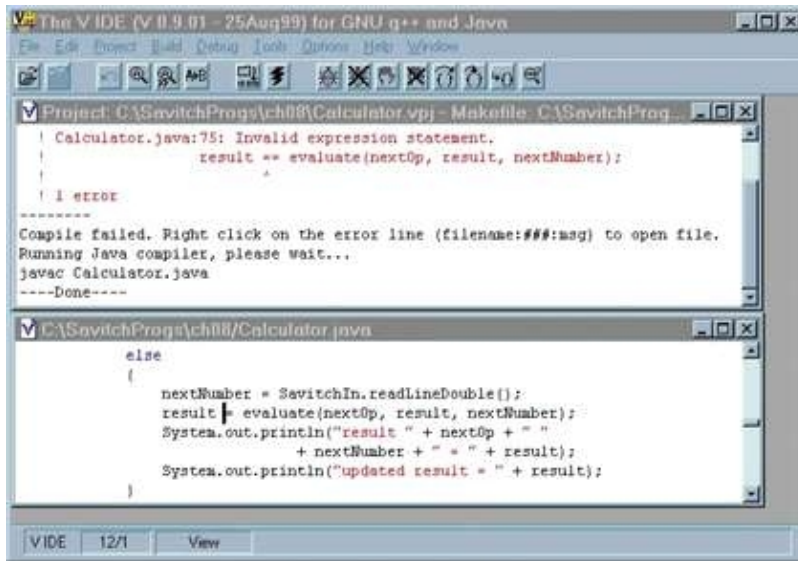


Java Syntax Error in VIDE

If there are too many errors, you can scroll the window to see them all. But best of all, you can right-click on error line, and VIDE will open the source file, and put the cursor on the line with the error.


(Note: you must put the cursor on a line that has the form "Filename:line#:message". If you click on a different line, VIDE will not open the source file.)

Once you've corrected the syntax error, you can simply click the build button again, and recompile the program. You continue this process until your program compiles successfully.



Java Syntax Error in VIDE - Fixed!

Running your program [top](#)

Once you have your program compiled, you need to run it with the Java interpreter. When you enter the **Tools:Run Project** menu command, or click the run icon , you will run your program with the interpreter set in the *Interpreter* pane of the project dialog. The java interpreter will run both console-type apps and GUI apps, but is really more suitable for running console-type apps. If you use awt or swing for GUI base apps, you should use javaw.

Java applets are handled differently. To view an applet, you must specify the HTML source in the *Main* pane of the project dialog. VIDE will then use the JDK appletviewer to run your applet. If you want to see your applet inside a browser window, you can edit the associated HTML file, and use **File:Send to Browser** to start your browser. Run your program by clicking the run button.

Debugging with VIDE [top](#)

Just as most program have syntax errors when they are first created, they have logic errors when you first run them. There are three main ways to find logic errors.

Most programmers neglect what is often the best way to find logic bugs – simply reading the source code carefully. While reading code isn't as much fun as running a debugger, it often is faster and easier. You should try to look at the code carefully first!

The second common way to find errors is to insert well placed print statements into your code. Print statements can easily show how far your code has gotten, and show values of variables. It also works well because you have to read the code to decide where to put the print statements.

The third way to find bugs is with the debugger. This way seems like the most fun, but often takes longer to find the bug than the more traditional code reading or trace print statements.

VIDE

The most common way to find bugs with a debugger is to set breakpoints. By setting a breakpoint, the debugger will stop your program when it tries to execute that statement. By setting breakpoints on statements just before and after the spot where the program seems to be going wrong, and then examining the values of variables after the breakpoint hits, you can often find out what is going wrong, and then fix the logic error in the code.

Besides examining variable values at a breakpoint, you can also step through the code a statement at a time. This will show you exactly which statements your program is executing.

As valuable as a debugger is for finding logic errors, it often is a real time waster. It is very easy to waste lots of time setting breakpoints where there is no error, or single stepping through perfectly correct code. Even so, there are many bugs that just can't be found without a debugger.

VIDE supports the standard GNU **`gdb`** and Sun **`jdb`** debuggers. The VIDE interface to the debuggers makes it far easier to debug your code, but is of minimalist design. The goal is to make using the native debuggers as easy as possible for casual users, while maintaining the full power of the debugger for experienced users. VIDE accomplishes this by showing a command window interface to the debugger. You can enter any native debugger command in this window, and thus have full access to all debugger features.

VIDE makes using the debugger easier by providing a dialog box with the most often used commands. And most importantly, VIDE will open the source file in an editor window and highlight the current execution line on breakpoints or steps. It is very easy to trace program execution by setting breakpoints, and clicking on the *Step over* or *Step into* debug dialog buttons. VIDE also allows you to inspect variable values by highlighting the variable in the source and right clicking the mouse.

A description of debug dialog commands and tool bar buttons is provided in the [VIDE Command Reference](#) section.

Debugging Java with `jdb` [top](#)

To effectively use `jdb`, you need a pretty good understanding of Java. One of the most confusing aspects of using `jdb` is threads. Many java apps, especially Java GUI apps, use threads, and this can lead to some confusion of just what you are debugging, and what will be displayed. This is just part of using Java.

To debug Java programs with `jdb`, you must first compile the program with debugging information. This is accomplished with the `-g` switch on the compile line. The current version of VIDE does not provide automatic generation of debug or release versions. The easiest way to define VIDE projects for both debug and release versions is to use the **Project:Save Project as...** command. First, define a release version of the project. Then, using that project as a template, change the switches as needed for your debug version, and save the project under a different name.

The full power of `jdb` is available in the debugger command window. You may enter any standard `jdb` command after the `>` prompt. In fact, there really is limited interaction between VIDE and `jdb`, mostly handling breakpoints. VIDE maintains its own list of breakpoints, which it keeps even if you start and stop the debugger. It is important that you use VIDE commands to set and delete breakpoints. If you enter breakpoints directly into the `jdb` command window, VIDE won't know about them, and won't highlight them in your source code.

Applets vs. Apps

Depending on whether you are debugging a Java app or a Java applet, you must start jdb differently. The standard way to start jdb from a command line is `jdb AppName`. To debug an applet, you would start jdb with `appletviewer -debug Applet.html`. What does this mean to you? It means when you want to debug an applet, you need to use the Java Project Editor (Project->Edit) and set the debugger name to "appletviewer -debug" on the Debugger tab. Normally, the debugger name is set to "jdb". If you created an applet project initially, this will be done automatically. Note that the appletviewer with the debugger is likely to take a *long* time to begin execution. Be patient.

Limitations with jdb

VIDE works quite well with jdb. In fact, it extends some of the capabilities by allowing you to delete all breakpoints at once.

VIDE Help System [top](#)

The VIDE help system has been designed to provide an excellent help environment for the programmer. The help files are supplied in HTML format. Different files are supplied depending on which VIDE distribution you have. The entire set of help files are available in a separate download from the ObjectCentral web site. V Help uses your default Web Browser to show the help files.

There are several help topics available from the VIDE help menu.



VIDE

This is the documentation for VIDE, including this file. It is included with all versions of VIDE.

Editor Command Set

This will display a dialog with a summary of the commands available for the current editor emulation. It is not an HTML file, but is internal to VIDE.

VIDE Help System

This will display the full VIDE Help System, which includes the best free documentation on GNU utilities, the GNU gcc compiler, C++, and HTML. The actual content for this help is available only as a separate download from the ObjectCentral web site. Clicking on this help before it is has been installed shows an

VIDE

HTML file with instructions for downloading the full version.

Win32 API

This item will bring up an HTML file telling how to download a Windows .HLP file that contains the WIN32 API reference. After you've installed that help file, this item will show it.

V GUI

This will show the V Reference Manual if you have downloaded the full V GUI distribution.

Java JDK

If you have installed the Sun JDK documentation, and set the Java Help path in the Options menu, this will bring up the entire Sun JDK help documentation in your browser.

HTML

This will bring up an HTML reference manual if it is installed. The HTML reference is normally included in the VIDE/Java distribution.

HTML – CSS

This will bring up an HTML reference manual if it is installed. The HTML reference is normally included in the VIDE/Java distribution.

No Warranty [top](#)

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler
All rights reserved.

Bruce E. Wampler, Ph.D.
bruce@objectcentral.com
www.objectcentral.com



[VIDE User Guide](#)

The Borland C++ Compiler 5.5

15 September 2000

- [Getting and Using the Free Borland C++ Compiler 5.5](#)
- [Setting up BCC 5.5](#)
- [Using Borland C++ with VIDE](#)
- BCC32 Quick Reference
 - ◆ [BCC32.EXE Switches](#)
 - ◆ [ILINK32.EXE – Switches](#)
 - ◆ [BCC32 Libraries](#)
 - ◆ Turbo Debugger Reference
 - ◇ [Command Line Switches](#)
 - ◇ [Keyboard Command Shortcuts](#)
 - ◇ [TD Help](#)
- [Other sites with help for BCC 5.5](#)
- [Help Improve VIDE for BCC](#)
- [Disclaimer](#)

This document is intended to help you use the free version of the Borland C++ compiler with VIDE. The Borland compiler is a good compiler, but the free version has some deficiencies. VIDE provides a development environment that makes using BCC 5.5 much easier. The free version is also a bit thin on its documentation. This situation has improved, and this document will help any BCC 5.5 user (with or without VIDE) use BCC 5.5 better. There is some important information about BCC 5.5 here that is very difficult to find elsewhere.

Getting and Using the Free Borland C++ Compiler 5.5 [top](#)

The Borland C++ Compiler 5.5 is available at <http://www.borland.com/bcppbuilder/freecompiler/>. You have to fill out a bunch of web forms, and eventually get to download the compiler. The download is about 8 Megabytes long.

Borland has made some service packs available to update the compiler. The latest full download on the Borland site will include the latest patches, but if you have a previous version of the compiler, you should apply the service packs.

At the same time you download the compiler, you should also download the debugger, Turbo Debugger, TD32. VIDE supports starting TD32 from inside VIDE. You must use TD to set breakpoints and view the source, but the interaction is as painless as possible.

Borland has also made extra documentation available. Most of what you need is included with the compiler, but documentation on the C and C++ libraries are not included with the free compiler. The most helpful files you need are in B5STD.ZIP found at <http://www.borland.com/techpubs/bcppbuilder/v5/updates/std.html>. That file is 7.5 MB, but is probably worth the download.

Once you download the compiler, the debugger, and the optional help files, you have to install them. The compiler and debugger downloads are self-extracting installers. It is probably a good idea to install them in the default location: `c:\borland\bcc55`. Once you've installed the compiler, you need to read the

README.TXT file. It tells important details for installation. These details are summarized in this document as well.

Setting up BCC 5.5 [top](#)

Configuration Files

The most important thing you need to do is set up two configuration files on the `\bin` directory. The Borland instructions don't make that location clear. Assuming you installed the compiler to the default locations, you need to create two files. The first, `C:\Borland\BCC55\bin\bcc32.cfg` should contain:

```
-I"c:\Borland\Bcc55\include"
-L"c:\Borland\Bcc55\lib;c:\Borland\Bcc55\lib\psdk"
```

The second, `C:\Borland\BCC55\bin\ilink32.cfg` should contain:

```
-L"c:\Borland\Bcc55\lib;c:\Borland\Bcc55\lib\psdk"
```

The purpose of these two files is to allow the compiler to find the standard system include and library files. Note that the Borland README.TXT leaves out the `\psdk` entry. If you leave that out, then the compiler won't be able to find all the standard Windows API files contained there.

Also note that you can add other entries to these files to change the default behavior of the compiler. For example, you might want to add `-wuse-` to `bcc32.cfg` to stop the compiler from issuing warnings about variables that are declared but never used. See the next section on specific switches recommended for VIDE.

Environment Path

In addition to these two configuration files, you need to add the compiler `\bin` directory to the PATH environment variable. On Windows9x, you edit the file `C:\autoexec.bat`. Simply add `c:\borland\bcc55\bin` to the PATH command. On NT, you use the system settings menu off the Start menu to change the PATH in the environment. Note that if you are using VIDE, you will need to have the VIDE directory on your path, too.

IMPORTANT WARNING!

The standard Windows header files included with Borland BCC 5.5 (e.g., `windows.h`) are set up to compile Windows applications for Windows 2000! This is NOT documented anywhere in the help files that come with BCC 5.5. There are some significant differences in some Win32 API calls for Win2K and earlier Win95, Win98, and WinNT 4.0 versions. Code compiled for Win2K (which is the default) will not run correctly on older versions of Windows. Code compiled for older versions will run on Win2K.

You should consider adding the following 2 lines to your `C:\Borland\BCC55\win32.cfg` file if you want code that works on Windows versions prior to Win2K!

```
-DWINVER=0x0400
-D_WIN32_WINNT=0x0400
```

These switches will be added to the compile line by VIDE, but it would be a good idea to include them in your `bcc32.cfg` file as well.

General Notes

Note that many switches can be negated by following it with a '-'. For example, '-v-' means no debugging information.

If you want to make any of these switches the default behavior, you can add them to the BCC32.CFG and ILINK32.CFG files in the /bin directory of the Borland command line tools.

Using Borland C++ with VIDE [top](#)

Using BCC32

VIDE hides most of the details of using the command line tools from you. However, underneath it all, the command line tools are still there. This section explains some of the details of using VIDE with BCC32.

Borland Configuration files

It is *essential* that you have the two compiler .cfg files set up in the \bin directory. The following files are suggested:

bcc32.cfg

```
-w
-I"c:\Borland\Bcc55\include"
-L"c:\Borland\Bcc55\lib;c:\Borland\Bcc55\lib\psdk"
```

The '-w' switch turns on warnings. You might want to refine the with some '-wxxx-' switches to suppress some of the warnings.

ilink32.cfg

```
-x
-L"c:\Borland\Bcc55\lib;c:\Borland\Bcc55\lib\psdk"
```

The '-x' switch turns off the map file. If you want to suppress incremental linking, you can add the '-Gn' switch.

VIDE Options

To use VIDE with the Borland compiler, you *MUST* set the path to the root of the compiler directory in the **Options**→**VIDE** dialog. You should set the `Compiler root:` value to the directory of the Borland compiler (not the \bin directory). If you installed BCC32 to the default directory, then this would be `c:\Borland\bcc55`. Note that if you've installed BCC32 to a directory with a space in its name, you need to enclose the path in quotation marks here. For example, `"c:\Program Files\Borland\bcc55"`. You should also select Borland BCC as the compiler in the Radio Button list.

Default Project Values

Depending on whether you generate GUI or a Console application, the VIDE project file sets some default values. These are visible in the **Project**→**Edit** project editor dialog.

The default compiler flags look like: `-P -O1 -v-`. The `-P` switch means C++ files, `-O1` is optimization for size, and `-v-` turns off debugging. Remember that you may have already set other switches in the `bcc32.cfg` file in the Borland compiler directory.

The linker flags line looks like: `-v- -Tpe -ap -c -limport32 -l$(BCC32RTLIB)`. These switches control the linker, and may change depending if you have a Console or GUI app. The last two values are the names of the run time libraries needed. `import32` is always needed, and the other, `BCC32RTLIB` is the It can be a static or dynamic version, and `cw32.lib` static version is used by default.

The linker also must include a startup object code file, which varies for GUI ("c0w32.obj") and console apps ("c0x32.obj"). There are also wide-char versions of these two startup libraries. You can override the defaults by changing the value of `BCC32STARTUP` in the Advanced tab of project editor.

Runtime Libraries

BCC32 comes with 4 runtime libraries. There are single threaded and multithreaded versions, and a static and dynamic version of each. The default library is "cw32.lib", the single-threaded static library. You can use the dynamic version of this library by changing the value of `BCC32RTLIB` in the advanced tab of the project editor to "cw32i" (no .lib, which is added automatically by VIDE). You also must either add the `-D_RTLDLL` define from the defines tab, or add the `-tWR` switch (`-tWCR` for console apps) to the compiler flags line, and recompile your program. You can do switch to the static multi-threaded library ("cw32mt") or dynamic library ("cw32mti") in a similar fashion.

DEF files

You may have some projects that require a .DEF file. If you need one, got to the Advanced tab in the project editor You will find the symbol `BCC32DEF` predefined with no value. Simply edit this entry to add the name of your .DEF file after the =. The file will then be used by `ILINK32`. See Borland's documentation for more information about using .DEF files.

Specifying Libraries

VIDE allows you to specify libraries to link with on the **Linker flags** line of the Names tab of the project editor. This line is used for linker flags, *and* the names of libraries you need to add. You can see the default `-limport32 -l$(BCC32RTLIB)` when you create a new project. You can add your own library names to this line, preferably *before* the `-limport32` entry. This `-l` syntax is not part of the Borland command line options, but is converted by VIDE to the form appropriate in the generated Makefile.

Using Turbo Debugger with VIDE

Since Turbo Debugger 32 (TD32) is a stand alone debugger, the integration with VIDE is somewhat limited. The main thing you can do is automatically launch TD32 with the debug button on the VIDE tool bar.

If you are building a V application, you will see the TD CPU window when it starts. This is because `WinMain` is in the V startup code, and not your application code. To view source, you must view a module. Use the TD menu commands **View**→**<Module** (or **F3**) to open the module (source file) with the code you want

VIDE

to debug. Non-V apps should start with the file with `main` or `WinMain` already shown.

Once you are running TD32, you use it to view the source code lines, set breakpoints, inspect variables, and all the usual debugging activities. When you find a bug, you then edit the source file with VIDE and recompile. Before you can recompile, you must quit TD32 (**Alt-X**). If you don't, you will get an error message from the linker.

When you then restart TD32, you should then see the message "Restart info is old, use anyhow?" If you answer "No", then all your old break points will be lost. If you answer yes, the breakpoints will still be there, and they get adjusted to the correct new line numbers if you've edited a file with breakpoints. Unfortunately, TD32 doesn't seem to know how to preserve open module windows.

If you are debugging a new or different project, you should answer "No" to the "Restart info" question the first time.

BCC32 Quick Reference

BCC32.EXE Switches [top](#)

switch	Switch Description
+filename	Use alternate configuration file named filename
@filename	Read compiler options from the response file filename
-3	Generate 80386 protected-mode compatible instructions. (Default for 32-bit compiler)
-4	Generate 80386/80486 protected-mode compatible instructions.
-5	Generate Pentium protected-mode compatible instructions.
-6	Generate Pentium Pro protected-mode compatible instructions.
-a	Default (-a4) data alignment; -a- is byte.
-an	Align to n. 1=byte, 2=word (2 bytes), 4=double word (default), 8=quad word (8 bytes), 16=paragraph (16 bytes)
-A	Use only ANSI keywords. (Extensions like the far and near modifier no longer recognized.)
-A- (Default)	Enable Borland C++ keyword extensions: near, far, huge, asm, cdecl, pascal, interrupt, _export, _ds, _cs, _ss, _es.
-AK	Use only KRkeywords.
-AT	Use Borland C++ keywords (Alternately specified by -A-)
-AU	Use UNIX V keywords. (Extensions like the far and near modifier no longer recognized.)
-b	Make enums always integer-sized. (Default: -b make enums integer size)
-B	Compiles assembly and calls TASM or TASM32. If you don't have TASM in your path, checking this option generates an error. Also, old versions of TASM might have problems with 32-bit generated assembler code.
-c	Compile source files, but does not execute a link command.

VIDE

-C	Turn nested comments on. (Default: -C- turn nested comments off.)
-d	Merge duplicate strings. (Default)
-Didentifier	Define identifier to the null string.
-Didentifier=string	Define identifier to string.
-efilename	Derives the executable program's name from filename by adding the file extension .EXE (the program name is then filename.EXE). filename must immediately follow the -e, with no intervening whitespace. Without this option, the linker derives the .EXE file's name from the name of the first source or object file in the file name list.
-Efilename	Use filename as the name of the assembler to use. (Default = TASM)
-f	Emulate floating point. (Default)
-f-	No Floating Point
-ff	Fast floating point. (Default)
-F	Uses fast huge pointers.
-Ff	Create far variables automatically.
-Ff=1	Array variable 'identifier' is near warning. (Default)
-Fm	Enables all the other -F options (-Fc, -Ff, and -Fs). Use this to quickly port code from other 16-bit compilers.
-gb	Stop batch compilation after first file with warnings (Default: -gb-).
-gn	Stop compiling after n messages. (Default: 255.)
-G	Optimize code for speed. (Default: -G- optimize code for size.)
-H	Generate and use precompiled headers. It might be called BC32DEF.CSM.
-H-(Default)	Does not generate and use precompiled headers.
-Hfilename	Sets the name of the file for precompiled headers
-H=filename	Set the name of the file for precompiled headers to filename.
-Hc	Cache precompiled headers. Use with -H, -Hxxx, -Hu, or -Hfilename. This option is useful when compiling more than one precompiled header.
-Hu	Use but do not generate precompiled headers.
-in	Make significant identifier length to be n, where n is between 8 and 250. (Default = 250)
-Ipath	Set search path for directories for include files to path.
-jb	Stop batch compilation after first file with errors. (default: off)
-jn	Errors: stop after n messages. (Default = 25)
-Ja	Expand all template members, including unused members.
-Jg	Generate definitions for all template instances and merge duplicates. (Default)
-Jgd	Generate public definitions for all template instances; duplicates result in redefinition errors.
-Jgx	Generate external references for all template instances.
-k	Turn on standard stack frame. (Default)

VIDE

-k-	Turn off standard stack frame. Generates smaller code, but it can't be easily debugged.
-K	Default character type unsigned. (Default: -K- default character type signed.)
-lx	Pass option x to the linker. More than one option can appear after the -l (which is a lowercase L).
-l-x	Disable option x for linker.
-Lpath	Set search path for library files.
-M	Instruct linker to create a full link map.
-npath	Set the output directory to path.
-O	Optimize jumps. (Default: on)
-O1	Generate smallest possible code.
-O2	Generate fastest possible code.
-Od	Disable all optimizations.
-Ox	There are a bunch of tiny optimizations, but it is probably only necessary to use -O1 and -O2 , so they are not covered here.
-OS	Pentium instruction scheduling. (Default: off: -O-S)
-p	Use Pascal calling convention. (This is a lowercase p.)
-pc	Use C calling convention. (Default same as -pc or -p-)
-pr	Use fastcall calling convention for passing parameters in registers.
-ps	Use stdcall calling convention (32-bit compiler only).
-P	Perform a C++ compile regardless of source file extension. (Default when extension is not specified. This is an uppercase P.)
-Pext	Perform a C++ compile regardless of source file extension and set the default extension to ext. This option is available because some programmers use .C or another extension as their default extension for C++ code.
-q	Quiet – suppress compiler banner.
-r	Use register variables. (Default)
-rd	Allow only declared register variables to be kept in registers.
-R	Include browser information in generated .OBJ files.
-RT	Enable runtime type information. (Default)
-S	Generate assembler source compiles the named source files and produces assembly language output files (.ASM), but does not assemble. When you use this option, Borland C++ includes the C or C++ source lines as comments in the produced .ASM file.
-tW	Make the target a Windows .EXE with all functions exportable. (Default)
-tWC	Make the target a console .EXE.
-tWD	Make the target a Windows .DLL with all functions exportable.
-tWM	Make a multithreaded application or DLL.
-tWR	Target uses the dynamic runtime lib. Can use -D_RTLDLL instead.
-tWCR	Target uses the dynamic runtime lib for CONSOLE apps.

VIDE

-T-	Remove all previous assembler options.
-Tstring	Pass string as an option to TASM, TASM32, or assembler specified with -E .
-u	Generate underscores for symbols. (Default)
-Uname	Undefines any previous definitions of the named identifier name.
-v	Turn on source debugging.
-vi	Turn on inline expansion (-vi- turns off inline expansion).
-V	Create smart C++ virtual tables. (Default) This means the .objs are compatible only with Borland tools. The V0 and V1 can apparently be used with other tools, but why?
-V0	Create external C++ virtual tables.
-V1	Create public C++ virtual tables.
-Vmd	Use the smallest representation for member pointers.
-Vmm	Member pointers support multiple inheritance.
-Vmp	Honor the declared precision for all member pointer types.
-Vms	Member pointers support single inheritance.
-Vmv	Member pointers have no restrictions (most general representation). (Default)
-Vx	Zero-length empty class member functions.
-w	Display warnings on.
-w!	Do not compile to .OBJ if warnings were found. (Note: there is no space between the -w and the !.)
-wmsg	Enable user define #pragma messages.
-w-xxx	Disable xxx warning message.
-wxxx	<p>Enable xxx warning message.</p> <ul style="list-style-type: none"> • amb – Ambiguous operators need parentheses. • amp – Superfluous with function. • asm – Unknown assembler instruction. • aus – 'identifier' is assigned a value that is never used. (Default) • bbf – Bit fields must be signed or unsigned int. • bei – Initializing 'identifier' with 'identifier'. (Default) • big – Hexadecimal value contains more than three digits. (Default) • ccc – Condition is always true OR Condition is always false. (Default) • cln – Constant is long. • cpt – Nonportable pointer comparison. (Default) • def – Possible use of 'identifier' before definition. • dpu – Declare type 'type' prior to use in prototype (Default) • dsz – Array size for 'delete' ignored. (Default) • dup – Redefinition of 'macro' is not identical (Default) • eas – 'type' assigned to 'enumeration'. (Default) • eff – Code has no effect. (Default) • ext – 'identifier' is declared as both external and static (Default) • hch – Handler for '<type1>' Hidden by Previous Handler for

'<type2>'

- hid – 'function1' hides virtual function 'function2' (Default)
- ibc – Base class '<base1>' is also a base class of '<base2>' (Default)
- ill – Ill–formed pragma. (Default)
- inl – Functions containing reserved words are not expanded inline (Default)
- lin – Temporary used to initialize 'identifier'. (Default)
- lvc – Temporary used for parameter 'parameter' in call to 'function' (Default)
- mpc – Conversion to type fails for members of virtual base class base. (Default)
- mpd – Maximum precision used for member pointer type type. (Default)
- msg – User–defined warnings . This option allows user–defined messages to appear in the IDE message window.
- nak – Non–ANSI Keyword Used: '<keyword>' (Note: Use of this option is a requirement for ANSI conformance.)
- ncf – Non–const function 'function' called for const object. (Default)
- nci – The constant member 'identifier' is not initialized. (Default)
- nod – No declaration for function 'function'
- nsf – Declaration of static function 'func(...)' ignored.
- nst – Use qualified name to access nested type 'type' (Default)
- ntd – Use '> >' for nested templates instead of '>>'. (Default)
- nvf – Non–volatile function function called for volatile object. (Default)
- obi – Base initialization without a class name is now obsolete (Default)
- obs – Identifier is obsolete. (Default)
- ofp – Style of function definition is now obsolete. (Default)
- ovl – Overload is now unnecessary and obsolete. (Default)
- par – Parameter 'parameter' is never used. (Default)
- pch – Cannot create precompiled header: header. (Default)
- pia – Possibly incorrect assignment. (Default)
- pin – Initialization is only partially bracketed.
- pre – Overloaded prefix operator 'operator' used as a postfix operator.
- pro – Call to function with no prototype. (Default)
- rch – Unreachable code. (Default)
- ret – Both return and return of a value used. (Default)
- rng – Constant out of range in comparison. (Default)
- rpt – Nonportable pointer conversion. (Default)
- rvl – Function should return a value. (Default)
- sig – Conversion may lose significant digits.
- stu – Undefined structure 'structure'
- stv – Structure passed by value.
- sus – Suspicious pointer conversion. (Default)
- ucp – Mixing pointers to different 'char' types.
- use – 'identifier' declared but never used.
- voi – Void functions may not return a value. (Default)
- xxx – Enable xxx warning message. (Default)

VIDE

	• zdi – Division by zero (Default)
-W	Creates a Windows GUI application. (same as -tW)
-WC	Creates a 32-bit console mode application. (same as -tWC)
-WD	Creates a GUI DLL with all functions exportable. (same as -tWD)
-WM	Make a multithreaded application or DLL. (same as -tWM)
-WU	Generates Unicode application. Uses -txxxx macros in tchar.h .
-x	Enable exception handling. (Default)
-xd	Enable destructor cleanup. (Default)
-xf	Enable fast exception prologs.
-xp	Enable exception location information.
-xp	Enable slow exception epilogues.
-X	Disable compiler autodependency output. (Default: -X- use compiler autodependency output.)
-y	Line numbers on.
-zAname	Code class set to name.
-zBname	BSS class set to name.
-zCname	Code segment class set to name.
-zDname	BSS segment set to name.
-zGname	BSS group set to name.
-zPname	Code group set to name.
-zRname	Data segment set to name.
-zSname	Data group set to name.
-zTname	Data class set to name.
-zX*	Use default name for X. For example, -zA assigns the default class name CODE to the code segment class.
-Z	Enable register load suppression optimization.

ILINK32.EXE – Switches [top](#)

ILINK32 objfiles, exefile, mapfile, libfiles, deffile, resfiles

@xxxx indicates use response file xxxx

switch	Switch Description
-ax	Specify application type (known x's follow)
-aa	Generate a protected-mode executable that runs using the 32-bit Windows API
-ap	Generate a protected-mode executable file that runs in console mode
-Ao:nnnn	Specify object alignment
-b:xxxx	Specify image base addr
-c	Case sensitive linking

VIDE

-C	Clear state before linking
-Dstring	Set image description
-d	Delay load a .DLL
-Enn	Max number of errors
-Gi	Generate import library
-Gl	Static package
-Gn	No state files
-Gpd	Design time only package
-Gpr	Runtime only package
-Gt	Fast TLS
-Gz	Do image checksum
-GC	Specify image comment string
-GD	Generate .DRC file
-GF	Set image flags
-GS	Set section flags
-H:xxxx	Specify heap reserve size
-Hc:xxxx	Specify heap commit size
-I	Intermediate output dir
-j	Specify object search paths
-L	Specify library search paths
-m	Map file with publics
-M	Map with mangled names
-q	Supress banner
-r	Verbose linking
-Rr	Replace resources
-s	Detailed segment map
-S:xxxx	Specify stack reserve size
-Sc:xxxx	Specify stack commit size
-Txx	Display time spent on link
-Txx	Specify output file type
-Tpd	Target a Windows .DLL file
-Tpe	Target a Windows .EXE file
-Tpp	Generate package
-Ud.d	Specify image user version
-v	Full debug information
-Vd.d	Specify subsystem version
-w-	Disable all warnings.
-wxxx	Warning control

	<ul style="list-style-type: none"> • def – No .DEF file • dpl – Duplicate symbol in lib • imt – Import does not match previous definition • msk – Multiple stack segment • bdk – using based linking in .DLL • dll – .EXE module built with .DLL extension • dup – Duplicate symbol • ent – No entry point • inq – Extern not qualified with __import • srf – Self–relative fixup overflow • stk – No stack
-x	No map

BCC32 Libraries [top](#)

The following is an incomplete listing of the main startup and runtime libraries included with BCC32. If you have more details, please send them, and I will include them here.

OBJ Files

C0D32.OBJ

32-bit DLL startup module (cod32w: wide-char version; cod32x: no exception handling)

C0S32.OBJ

Unknown

C0W32.OBJ

32-bit GUI EXE startup module (c0w32w: wide-char)

C0X32.OBJ

32-bit console-mode EXE startup module (c0x32w: wide-char)

FILEINFO.OBJ

Passes open file-handle information to child processes. Include this file in your link to pass full information about open files to child processes created with exec and spawn. Works with the C++ runtime library to inherit this information.

GP.OBJ

Prints register-dump information when an exception occurs.

WILDARGS.OBJ

If you want wild-card expansion for you console-mode applications, then you should also link in this file when you link your console-mode application. It apparently doesn't work for GUI apps. It does the normal DOS-like wild card expansion and passes them to argv and argc of your main.

LIB Files

CW32.LIB

32-bit single-threaded static library

CW32I.LIB

32-bit single-thread, dynamic RTL import library for CW3250.DLL. To use this import library, you must compile your programs with either -D_RTLDLL or -tWR options to the compiler. This probably applies to the other "i" libs as well.

CW32MT.LIB

32-bit multithread static library

CW32MTI.LIB

Import lib for 32-bit multithread dynamic RTL import library for CW3250MT.DLL

IMPORT32.LIB

- 32-bit import library
- dxextra.lib***
DirectX static library
- inet.lib***
Import lib for MS Internet DLLs
- noeh32.lib***
No exception handling support lib
- ole2w32.lib***
Import lib for 32-bit OLE 2.0 API
- oleaut32.lib***
Unknown
- uuid.lib***
GUID static lib for Direct3d, DirectDraw, DirectSound, Shell extensions, DAO, Active Scripting, etc.
- wininet.lib***
Unknown
- ws2_32.lib***
Import lib for WinSock 2.0 API

Turbo Debugger Commands [top](#)

Command Line Switches

Command Line Syntax:
 TD32 [options] [program [arguments]] -x- = turn option x off

switch	Switch Description
-as	-ar (NT only) Attach to running process: s=stop, r=run, id = #
-ae	(NT only) Post-mortem event handle, decimal event handle = #
-c<file>	Use configuration file <file>
-h,-?	Display this help screen
-ji,-jn,-jp,-ju	State restore: i=Ignore old,n=None,p=Prompt old,u=Use old
-l	Assembler startup
-p	Use mouse
-sc	No case checking on symbols
-sd<dir>	Source file directory <dir>
-t<dir>	Start with current directory <dir>

Keyboard Command Shortcuts [top](#)

Key	Command
<i>File Menu</i>	
Alt-X	Quit
<i>Edit Menu</i>	

VIDE

Shift-F3	Copy
Shift-F4	Paste
<i>View Menu</i>	
F3	Open Module (source file)
<i>Run Menu</i>	
F9	Run
F4	Go to cursor
F7	Trace into
F8	Step over
Alt-F9	Execute to...
Alt-F8	Until return
Alt-F4	Back trace
Alt-F7	Instruction trace
Ctrl-F2	Program reset
<i>Breakpoints Menu</i>	
F2	Toggle breakpoint...
Alt-F2	At breakpoint...
<i>Data Menu</i>	
Ctrl-F4	Evaluate/modify...
Ctrl-F7	Add watch...
<i>Window Menu</i>	
F5	Zoom
F6	Next
Tab	Next pane
Ctrl-F5	Size/move
Alt-F3	Close
Alt-F6	Undo close
Alt-F5	User screen
<i>Help Menu</i>	
Shift-F1	Help Index
Alt-F1	Previous topic
xx	xx
xx	xx

TD Help [top](#)

The following list contains all the topics covered by TD help. Those especially relevant to C and C++ programming are presented here.

• **Help on Help**

Welcome to Turbo Debugger's help facility. Use the following keys to move around:

Arrows – Move to topic	Enter – Go to topic
PgDn– Next screen	F1 – Help index
Alt–F1 – Previous screen	Ctrl–PgUp – First screen

- **Assembler**
- **Assembler constants**
- **Assembler expressions**
- **Assembler operators**
- **Assembler strings**
- **Assembler variables**
- **Auto–variables**

Auto–variables are defined only within the function or block in which they are declared, and they exist only when the function they are in is called. They don't retain their values from one function call to the next.

Turbo Debugger handles auto–variables by emulating the current language's variable scoping rules. That is, whenever the program is stopped inside a function, the auto–variables for that function are added to or supersede the static and local symbols already known. They can be used in expressions exactly as they would be used in a statement inside the function.

The bottom pane of the Variables window shows the current value for all auto–variables in the active function.

• **Breakpoints**

You use breakpoints to specify something you want done when your program has run to a specific source line number or address.

You can set a breakpoint to stop your program at any source line or address. Position the cursor on the desired line and press F2. Press F2 again to clear the breakpoint.

You can also set a breakpoint by clicking the mouse in one of the first two columns of the line that you want to set the breakpoint on. Turbo Debugger's breakpoints perform all the same jobs as the breakpoints, watchpoints, and tracepoints of other debuggers.

A breakpoint can be instructed to perform one of the following actions:

Break	Stop your program
Log	Log the value of variables
Execute	Execute an expression

You can set a breakpoint to execute on one of the following conditions:

Always	Every time it is encountered
Expression true	Only when an expression is true
Changed memory	Only when a memory area changes
Hardware	When a hardware breakpoint occurs

Breakpoints can also be qualified by setting a Pass Count that specifies how many times the breakpoint must be passed over before being activated.

A breakpoint can also be set at a global address, which means that it occurs on every source line or instruction address. This allows you to watch the value of a variable as each line is executed, or to stop when a variable or area of memory changes value.

- **Byte lists**

- **C constants**

Constants can be either floating point or integer. A floating-point constant contains a decimal point and can be in decimal or scientific notation; for example,

```
1.234    4.5e+11
```

Integer constants are decimal unless you use one of the C conventions for overriding the radix:

Format	Radix
digits	Decimal
0digits	Octal
0xdigits	Hexadecimal
0xdigits	Hexadecimal

Ending any integer constant with "l" or "L" makes it into a long constant.

- **C Debugging tips**

This TD help section is meant to aid the less experienced programmer in finding bugs in C programs. Some examples of common pitfalls are given, along with suggestions on how to resolve and avoid them. See the TD help for details.

- **C expressions**

The full C expression syntax is supported, which allows you to evaluate any C expression that the compiler accepts.

Variables can be changed using the full range of C assignment operators or the increment and decrement operators (++ and --).

A C expression can contain the following elements: Operators, Strings, Variables, Constants

- **C operators**

Turbo Debugger supports all the C operators and one additional special operator, ::, which has the highest priority. You use it to generate constant far-memory pointers to characters; for example,

```
0x1234::0x5678
```

The primary expression operators

```
()  []  .  ->
```

have the next highest priority and group from left to right.

The unary operators

```
*  !  ~  ++  --
```

VIDE

have priority less than the primary operators but greater than the binary operators and group from right to left.

The binary operators are listed here in decreasing priority. Operators on the same line have the same priority.

```
highest * / %
        + -
        >> <<
        < > <= >=
        == !=
        ^
        |
        &
lowest  ||
```

The single ternary operator has a priority less than any of the binary operators:

```
?:
```

The assignment operators all have the same priority below the ternary operator and group from right to left:

```
= += -= *= /= %= >>= <<= = ^= |=
```

• C strings

C strings in Turbo Debugger use exactly the same syntax as in C source files: a series of characters between double quotes (" "). All characters are inserted literally except for the backslash, which is used to enter special character codes into the string.

If a backslash is followed by any character except one of those listed here, that character is inserted into the string unchanged. This allows the string delimiter double-quote character (") to be entered by typing \".

The following escape sequences can be used:

Sequence	Value	Character
-----	-----	-----
\\a	0x07	Bell
\\b	0x08	Backspace
\\f	0x0c	Formfeed
\\n	0x0a	Newline (line feed)
\\r	0x0d	Carriage return
\\t	0x09	Horizontal tab
\\v	0x0b	Vertical tab
\\\\	\\	Backslash
\\xnn	nn	Hex byte value
\\nnn	nnn	Octal byte value

• C variables

Variables are names for data items in your program. A C variable name starts with a letter (a – z, A – Z) or the underscore character (_). Subsequent characters in the symbol can be digits (0 – 9) as well as these characters.

VIDE

Normally, a variable name obeys the C scoping rules, with auto-variables overriding static variables of the same name, which override globals. You can override this scoping if you wish to access variables in other scopes.

• Command line options

Here's the general form of the DOS command line used to start Turbo Debugger:

```
td [options] [program_name [program_args] ]
```

Items enclosed in brackets are optional. "Options" are described below. "Program_name" is the name of the program to debug. If no extension is supplied, .EXE is presumed. "Program_args" are any arguments required by the program being debugged. Do not place debugger options here; they should come before the name of the program being debugged.

Example command lines:

Command	Action
===== td	===== Start the debugger with no program, default options
td -r prog1 a	Start the debugger with -r option, load program "prog1" with one argument "a"
td prog2 -x	Start the debugger with default options, load "prog2" with one argument "-x".

An option consists of a hyphen, followed by one or more letters and a possible text or numeric argument. Options override default configuration settings from the configuration file. To disable an option set in the configuration file, follow the option with another hyphen, for example,

```
td -p- myprog arg1 arg2
```

• Configuration file

The configuration file allows command-line options to be set automatically each time the debugger is started. The configuration file is searched for first in the current directory, then in the directory containing the Turbo Debugger executable program. This allows you to have a system-wide configuration in the PATH and custom configurations for individual projects.

Options on the command line override the corresponding setting in the configuration file.

• Cursor in List panes

In a List pane items can be selected in one of two ways:

- ◆ Use the cursor keys to position the highlight bar.
- ◆ If the List pane has a blinking cursor, start to type the letters making up an entry. The highlight bar moves to the first item that starts with the letters typed so far; this allows you to select items with a minimum of typing. This is referred to as "incremental matching."

The following keys can be used to move the highlight bar in a List pane:

Up	Move up one item
Down	Move down one item
PgUp	Move up one pane height

VIDE

PgDn	Move down one pane height
Ctrl-PgUp	Move to first item in list
Ctrl-PgDn	Move to last item in list
Esc	Cancel incremental search
Enter	Select item in list
a - z	Incremental match on list items

• Cursor in Text panes

Use the following keys to control the cursor position in Text panes:

Up	Move cursor up one line
Down	Move cursor down one line
Right	Move cursor right one character
Left	Move cursor left one character
Ctrl-Right	Move right one word
Ctrl-Left	Move left one word
PgUp	Move up one page
PgDn	Move down one page
Ctrl-Home	Go to top line of window
Ctrl-End	Go to bottom line of window
Ctrl-PgUp	Go to first line in file
Ctrl-PgDn	Go to last line in file
Home	Go to beginning of current line
End	Go to end of current line

You can mark a block of text by pressing Ins, then moving the cursor within the same line. Some prompts use this marked block as their default response entry.

- CPU Flags
- Disassembler
- DOS commands
- Error messages
- Examining Data

You can examine data items with the Inspect option on the Data menu. You can inspect any expression that evaluates to a memory pointer, both simple variable names and complicated address expressions.

You can also examine data items by pointing to a variable name in a Text or List pane and specifying the Inspect option from the speed menu for that pane. In a Text pane, this command works if the cursor is anywhere within the variable name you wish to inspect. You can also use the Ins key to mark an expression to inspect.

• Expressions

You can evaluate an expression using any one of the languages supported by Turbo Debugger:

C	Pascal	Assembler
---	--------	-----------

Use the Options/Language command to select which language is used to interpret expressions you type in. Normally, Turbo Debugger uses the language that you used to write the code at the current program address.

When you display an expression, you can enter an optional format control string that controls how the result is displayed.

• **Format Control**

When you supply an expression to be displayed, Turbo Debugger displays it in a suitable format based on what type of data it is. If you wish to change the default display format for an expression, place a comma at the end of the expression, then supply an optional repeat count followed by an optional format letter. You can supply a repeat count only for pointers or arrays. If you use a format control on the wrong data type, it has no effect.

- c Displays a character or string expression as raw characters. Normally, non-printing character values are displayed as some type of escape or numeric format. This option forces the characters to be displayed using the full IBM display character set.
- d Displays an integer as a decimal number.
- f# Displays a floating-point format with the specified number of digits. If you don't supply a number of digits, as many as necessary are used.
- m Displays a memory-referencing expression as hex bytes.
- md Displays a memory-referencing expression as decimal bytes.
- p Displays a raw pointer value, showing segment as a register name if applicable.

• **Function Keys**

There are function keys for many commonly used functions. The regular function keys invoke the following commands:

- | | |
|----------------|----------------------|
| F1 Help | F2 Toggle Breakpoint |
| F3 View Module | F4 Go to Cursor |
| F5 Zoom Window | F6 Next Window |
| F7 Trace Into | F8 Step Over |
| F9 Run | F10 Command menu |

The Alt function keys give you access to other commands you use fairly often.

- | | |
|---------------------|----------------------|
| Alt F1 Last help | Alt F2 Breakpoint At |
| Alt F3 Close Window | Alt F4 Back trace |
| Alt F5 User Screen | Alt F6 Undo Close |
| Alt F7 Instr. Trace | Alt F8 Until Return |
| Alt F9 Execute To | Alt F10 Speed menu |

• **Global variables**

Global variables are those defined as being accessible to all modules in the program. You can view the global variables in a program by creating a Variables window. The global variables appear in the upper pane of that window.

• **Help line**

The bottom line of the screen always displays help about some keys specific to the current environment. This line changes depending on where you are.

VIDE

Normally, it shows the most commonly used function key commands. Press and hold down the Alt key and it shows the commonly used Alt function keys.

- **Hierarchy window**

The Hierarchy command opens an Object Hierarchy window that shows all the Pascal or C++ object types, both in alphabetical order and as a tree showing ancestor/descendant relationships.

- **Inspectors**

You'll probably use Inspectors more than any other sort of window. You can create an Inspector in one of two ways:

- ◆ Use the Inspect speed menu command in a pane by typing Ctrl-I or Alt-F10 I.
- ◆ Use the Data Inspect command from the main menu.

An Inspector shows the contents of a data structure in the program you are debugging. It also allows you to modify individual variables or members of a data structure.

The most common way to use Inspectors is to open one by placing the cursor on what you want to inspect and pressing Ctrl-I. After you've examined the data item, press the Esc key to remove the Inspector window.

If you have several Inspectors on the screen, the Window Close command (or its equivalent, the Alt-F3 key) removes them all at once.

- **Interrupt key**

The Interrupt key can be pressed at any time your program is running. Turbo Debugger immediately regains control, allowing you to examine the state of the program.

Usually the Interrupt key is the Ctrl-Break key combination; you can change this to another key by using the TDINST customization program.

- **Logging events**

Turbo Debugger's breakpoints perform all the same jobs as the breakpoints, watchpoints, and tracepoints of other debuggers.

A breakpoint can be instructed to perform one of the following actions:

Break	Stop your program
Log	Log the value of variables
Execute	Execute an expression

- **Macros**

The Options Macros command lets you set and clear macros assigned to different keys.

Select one of the following commands to learn more about it:

Create	Remove
Delete All	Stop Recording

You can use macros to record command sequences that you often repeat, as well as to record commands from the start of a session. This is very useful when you must keep restarting and getting back to a certain place in your program.

• **Main Menu**

The menu bar appears on the top line of the screen. You can return to this menu at any time by pressing the F10 hot key. A combination of pull-down and pop-up menus gives you easy access to all the menu commands and options. The menu bar has the following commands:

System	Breakpoints
File	Data
View	Window
Run	Options
Edit	

Access the System menu in the top left corner by pressing Alt-space.

◆ System

The System menu lets you invoke commands that affect the entire screen (desktop).

The Repaint desktop command redisplay the entire Turbo Debugger screen. This is useful if screen swapping has been disabled and the debugger screen has become corrupted with output from the program you are debugging.

Use the Options/Display options command to set how screen updating is done.

The Restore Standard command restores the window layout to the way it was when the debugger was first started.

The About command shows the sign-on copyright screen that identifies the version of Turbo Debugger that you are using.

◆ File

The File menu has a number of options that deal with operations external to Turbo Debugger, such as loading programs to debug, executing DOS commands, and returning to DOS.

The Open command loads a program to debug from disk. You are prompted for the name of the program to debug. You don't need to supply the .EXE or .COM extension.

You can supply any arguments to the program by specifying them after the program name, just as you would on the DOS command line.

The Change Dir command lets you set a new current drive and/or directory by prompting you for the new drive and/or directory name.

The prompt is initialized to the current drive and directory, so you can also use this command to examine the current drive and directory settings.

The File Get Info command displays a window that shows you the current state of the program you are debugging. This includes:

- ◇ the program name
- ◇ where and why your program is stopped
- ◇ main memory usage
- ◇ interrupts your program has intercepted
- ◇ the DOS version

VIDE

- ◇ Hardware/software breakpoints indicator
- ◇ the current date and time

The Attach command lets you debug a program that is already running and was not started by Turbo Debugger. You supply the process ID of the process that you wish to debug. Turbo debugger then interrupts the process just as if a breakpoint had been encountered.

The Quit command returns control to DOS. Alt-X is the hot key that returns to DOS from any point while Turbo Debugger is running.

The memory for your program image is freed, and any open file handles are closed. If your program has allocated memory blocks from DOS, Turbo Debugger frees these as well.

◆ Edit

The Edit menu has options that let you put things onto the clipboard and to place items on the clipboard into the current window.

The Copy command copies the currently highlighted item or position in a window to the clipboard. You can then use the Paste command to put the item somewhere else.

The Paste command copies an item from the clipboard into the current context in a window. You are given a list of clipboard items that are able to be used in the current context. To see the entire contents of the clipboard, use the View/Clipboard main menu command.

The Copy to log command copies the currently highlighted item or position in a window to the Log window.

The Edit Dump Pane to Log command writes the contents of the current pane to the Log window. This can be useful when you want to compare the values of some variables at two different places in the program. You can dump the contents of the Watches window or some Inspectors, then execute some more of your program and compare the values recorded in the log with the current values.

◆ View

The View menu commands open windows that "view" part of the program you are debugging. Use the Another option in this menu to open another instance of a Dump, File, or Module window that you already have displayed on the screen.

The Breakpoints command opens a Breakpoints window that shows a list of the current breakpoints, their conditions, and what they do when they're triggered. In Turbo Debugger, breakpoints combine the functions of breakpoints, watch points, and trace points.

The CPU command opens a CPU window that shows disassembled instructions, a hex data dump, the CPU registers and flags, and the stack contents. You can also patch code using the built-in assembler. If there is already a CPU window on the screen, you are positioned in that window.

The Dump command opens a Dump window showing the contents of an area of memory that you specify. You can view the data as raw hex bytes or in any of the supported number formats, including 32-bit integers and four types of floating-point numbers. This window works the same as the Data pane in a CPU window. This window is useful if you just want to look at an area of memory, and don't care about the rest of the CPU state that the CPU

window displays.

The Execution history command opens an Execution history window that shows a list of instruction that can be undone, and a list of keystroke checkpoints that you can recover to. This lets you "go back in time" if you step beyond the location where a problem occurs.

The File command opens a File window that shows the contents of a file that you specify. You are prompted for the file name; you can enter a wildcard specification and select a file from the displayed list. You can also edit the contents of the file in hex or ASCII text. If there is already a File window on the screen, the file you selected is displayed in that window. To open another File window, use the View/Another main menu command.

The Hierarchy command opens an Object Hierarchy window that shows all the Pascal or C++ object types, both in alphabetical order and as a tree showing ancestor/descendant relationships.

The Log command opens a Log window that shows a scrolling log of messages from the following events:

- ◇ Breakpoints that log an expression
- ◇ Program trace, step, and breakpoint addresses
- ◇ User comments
- ◇ Window pane dumps

The contents of the Log window can also be written continuously to disk.

The Clipboard command opens a Clipboard window showing the items copied to the clipboard. You can examine clipboard items, delete them, or open a Window to show you the contents of an item. Use the Edit/Copy command to put text, data and addresses onto the clipboard. Use the Edit/Paste command to place clipboard items into prompts or windows. The type of data copied to the clipboard depends on which window or pane it is clipped from.

The Module command opens a Module window that shows a source file for the module. You can select a module to view from the list of modules in the program. If there is already a Module window on the screen, the module you selected is displayed in that window. If you want to open another Module window, use the View/Another main menu command. F3 is the hot key for this command. The Module window title shows the name of the module you are viewing, the current source file for the module, and the line number that the cursor is on. If the source file has been changed since the module was compiled and linked into your program, the message (modified) appears before the current line number. This warns you that your source file might be out of sync with the program you are debugging and that the cursor might not always appear on the correct line of source code.

The Numeric Processor command opens a Numeric Processor window that shows the contents of the 80x87 stack registers, the control bits, and the status bits. You can modify the floating-point registers and the control and status fields of the numeric processor.

The Registers command opens a window that shows the current contents of the CPU registers and flags. You can examine or change their values. This window works the same as the Register pane in a CPU window. It is useful if you just want to look at the CPU registers, and don't care about the rest of the CPU state displayed in the CPU window.

VIDE

The Stack command opens a Stack window that shows the currently active calling stack. All functions starting from the call at the start of the program are shown, along with their arguments. From this window, you can also examine the variables inside a particular instance of a function on the stack; this is useful, for example, if you are using a recursive function.

The Variables command opens a Variables window that shows a list of symbols and their values for the current module and function, as well as a list of your program's global symbols and their values.

The Watches command opens a Watches window displaying the value of variables or expressions in your program that you specified using either the Data Add Watch main menu command or the Watch speed menu command in a Module window.

◆ Run

The Run menu commands execute your program. If Screen Updating is enabled, your program screen is restored before execution continues.

The Trace Into command executes a single instruction or source line. If the current window is not a CPU window, the next source line is executed. If the current window is a CPU window, a single instruction is executed. Some instructions cause the CPU to execute the following instruction as well as the current instruction. Among these are instructions such as loading of segment registers, for example, `MOV SS,AX`. The F7 key is the hot key that executes this command.

The Step Over command executes a single instruction or source line and skips over any procedure calls. If the current window is not a CPU window, the next source line is executed. If any part of the source line contains function calls, they are executed without stopping. If the current window is a CPU window, a single instruction is executed. If the current instruction is a `CALL` instruction, the entire subroutine is executed without interruption, and the program stops at the instruction following the `CALL` instruction. Instructions with a `REP`, `REPNE`, or `REPZ` prefix are also executed without interruption. The F8 key is the hot key that executes this command.

The Go to Cursor command executes your program until it reaches the currently highlighted source or instruction line. The current window must be a CPU or a Module window so that the location to execute to can be determined. The F4 key is the hot key that executes this command.

The Execute To command executes your program and stops at the specified location in the program. You are prompted for the program address at which to stop. You can enter an address in any of the valid address formats. `Alt-F9` is the hot key that executes this command.

The Program Reset command reloads the current program from disk. This command is useful if you've run the program too far during a debugging session and need to restart execution at the start of the program. This command is necessary, since you usually can't restart a program simply by starting execution at the beginning of the program. A full reload from disk reinitializes data items that the program expects to contain a particular value when the program is first started. `Ctrl-F2` is the hot key for this command.

The Run Arguments command lets you set or change the command-line arguments for the program you are debugging. Use this command if you are debugging a program that requires

VIDE

one or more command–line arguments, but you entered either no arguments or erroneous ones with the File/Open command.

The Run/Wait for child main menu command lets you specify whether Turbo Debugger waits for the program you are debugging to stop before letting you issue additional commands. If you set it to No, Turbo Debugger lets you issue more commands after you run the program you are debugging. This lets you examine data items in your program while it is executing. However, you can't examine the CPU registers as the program is running – you see their values as of when your program last stopped. Use the Run/Next pending status command to retrieve status information from your program.

The Animate command is a self–repeating Trace Into command. Instructions or source lines are executed continuously until a key is pressed. Turbo Debugger's display changes to reflect the current program state between each trace, which allows you to watch the flow of control in your program. You are prompted for the rate at which to step instructions.

The Run command executes your program continuously until either a breakpoint is encountered, the program is interrupted with the Interrupt key, or the program terminates. The F9 key is the hot key that executes this command.

The Until Return command executes the program being debugged until the current function or procedure returns to its caller. This is useful if you have accidentally issued a Trace Into command when you really wanted to do a Step Over command, or when you have stepped through enough of a function to determine that it is working correctly, and you wish to execute the rest of it without interruption. Alt–F8 is the hot key that executes this command.

The Instruction Trace command executes a single machine instruction. After using the command, you are usually left in a CPU window. Use this command when you want to

- ◇ trace into an interrupt call in the CPU window.
- ◇ trace into a function in a module that was not compiled with debug information.
- ◇ watch each instruction execute that makes up a source line.

Alt–F7 is the hot key for this command.

The Back trace command undoes the last instruction or source statement. The processor and memory state are restored to the way they were before the instruction or source line was traced over. The Alt–F4 key is the hot key for this command. The history pane of the Execution history window contains a list of saved steps and allows you to control when stepping and tracing information is saved.

The Next pending status command lets you retrieve a status message from a process that is executing asynchronously. If you use the Wait for child command to specify that the program you are debugging can run in the background while Turbo Debugger is running, the status display in the top right corner of the screen changes to say PENDING when a process wants to report something, such as a breakpoint being hit or the process terminating.

◆ Breakpoints

The Breakpoints menu commands let you set and clear breakpoints.

The Toggle command sets or clears a breakpoint at the currently highlighted address in a Module window or CPU window Code pane. Your program stops every time it reaches a line

VIDE

where you have set a breakpoint. You can modify the behavior of the breakpoint by opening a Breakpoint window. This lets you set a pass count or a condition that must be true before the breakpoint happens, and also define what to do when the breakpoint is triggered. The F2 key is the hot key that executes this command. You can also click the mouse in the first two columns.

The At command sets a breakpoint at a specific location in your program. You are prompted for the address at which to set the breakpoint and for any options. Alt-F2 is the hot key that executes this command.

The Changed Memory Global command allows you to set a breakpoint that occurs when an area of memory changes value. You are prompted for the memory area to watch. Your program executes more slowly if you set any of these breakpoints. If you are running on an 80386 or later processor, make sure you've installed the hardware device driver supplied on the distribution disk; this speeds up global breakpoints. If you have a hardware debugger board, see if the vendor supplies a device driver that lets it work with Turbo Debugger.

The Expression True Global command allows you to set a breakpoint that occurs when the value of an expression becomes true. You are prompted for the expression that must become true in order for your program to stop. Your program executes more slowly if you set any of these breakpoints.

The Hardware breakpoint command lets you quickly and easily set a general purpose hardware breakpoint. The dialog box lets you set the type of hardware breakpoint, exactly as if you had issued the Hardware options command in the Breakpoint window speed menu.

The Delete All command removes all the breakpoints from your program. Use this command when you want to continue debugging your program but no longer want it to stop at any breakpoint locations you've previously set.

◆ Data

The Data menu commands allows you to examine variables and memory areas in your program. You can also evaluate Expressions and change their values.

The Inspect command opens an Inspector to show the value of a variable or a memory-referencing expression. You are prompted for the variable or memory-referencing expression to inspect. If the cursor was in a Text pane when you issued this command, the prompt is initialized to contain the variable under the cursor, if any. If you have marked an expression using the Ins key, the prompt is initialized to the marked expression.

The Evaluate/Modify command evaluates an arbitrary expression. Enter an expression followed by an optional format control string that's separated from the expression with a comma (.). The value appears in the middle pane. You can move to this pane to scroll long value displays or error messages. If the result is changeable, you can change the expression's value by moving to the bottom pane (press the Tab key), typing a new value, and pressing Enter. If the cursor was in a Text pane when you issued this command, the prompt is initialized to contain the variable under the cursor, if any. If you have marked an expression using the Ins key, the prompt is initialized to the marked expression.

The Data Add Watch command places an expression or variable on the watch list displayed by the Watches window. If the cursor was in a Text pane when you issue this command, the

VIDE

prompt is initialized to contain the variable under the cursor, if any. If you have marked an expression using the Ins key, the prompt is initialized to the marked expression.

The Data Function return command lets you examine the value that is about to be returned by the current function. You can only issue this command just before the function is about to return to the function that called it.

◆ Options

The Options menu commands allow you to adjust a number of options that have a global effect on the behavior of Turbo Debugger.

The Options Language command lets you specify how Turbo Debugger interprets expressions you type in. You can select from one of the languages that Turbo Debugger supports.

The Options Macros command lets you set and clear macros assigned to different keys.

- ◇ Create – The Create command lets you assign a new sequence of keystrokes to any key on the keyboard. Alt = is the hot key for this command. You are prompted to press the key you wish to assign a macro to. Then you can enter the keystrokes that make up the macro, which is acted upon just as if you typed them without recording a macro. End the macro definition by pressing either Alt – or the key you are assigning the macro to, or issuing the Options Macros Stop Recording main menu command.
- ◇ Delete All – The Delete all command deletes all keystroke macros. Use this command when you wish to restore all keys to their original functions and free the macro memory for a new set of keystroke macros.
- ◇ Remove – The Remove command lets you remove a sequence of keystrokes assigned to a key. This returns the key to its original function. You are prompted to press the key for the macro you wish to delete.
- ◇ Stop Recording – The Stop Recording command stops keystroke remembering for the macro you're presently recording. If you use this menu command, it will be recorded as part of the macro. Use one of the following single-key stop methods to avoid this:

- * Alt - (hyphen).
- * Pressing the key that you are assigning the macro to.

Assigning no keystrokes to a key makes it perform no action. To restore a key to its original function, use the Remove command. You can use macros to record command sequences that you often repeat, as well as to record commands from the start of a session. This is very useful when you must keep restarting and getting back to a certain place in your program.

The Display options command lets you adjust the appearance of Turbo Debugger's display screen.

The Options Path for Source command lets you set where Turbo Debugger looks for the source files that make up your program. Source files are searched for first where the compiler found them, then in each directory specified by this command or the –sd command line option, then in the current directory, and finally in the directory that contains the program you are debugging.

The Save Options command brings up a dialog that lets you select which part of your current configuration you wish to save to disk and the file name to use.

The Restore Options command loads a configuration file from disk. You must have previously saved the configuration file by using the Save options command or the customization program to create the configuration file. Depending on which option of the Save options command you used, the configuration can contain only macros, only window layouts, or an entire configuration.

◆ Window

The Window menu commands let you perform certain operations on the windows on screen with the following commands:

Zoom	Size/move	Undo	Close
Next	Iconize/restore		
Next Pane	Close	User	screen

At the bottom of the list appears a list of all the windows open on the screen. Select one of these to go directly to that window. Use the View menu commands to create new windows. The F5 key zooms the current window and the F6 key takes you to the next window.

• **Modifying data**

There are several ways to modify program data:

- ◆ Use the Data Evaluate/Modify command and type in a new value for an expression.
- ◆ Use the Data Evaluate/Modify command and evaluate an expression that has side effects, such as an assignment statement or one that uses the C ++ or -- operators on a pointer variable.
- ◆ Use the Change command in the speed menu for the Data pane of a CPU window. This is good for examining and modifying data as a raw hex dump.

• **Moving windows**

The Size/move command lets you change the size and/or position of the current window. Once you have issued the command, use the cursor keys to move the window and Shift plus the cursor keys to resize it. Pressing the Ctrl-F5 key is the hot key for this command.

• **Operator precedence**

• **Panes**

Panes are the individual display areas that make up each window. Each pane is associated with a speed menu of commands unique to that pane. There are two types of panes: List panes and Text panes. List panes show an alphabetically ordered list of items from which to select. Text panes display the contents of a disk file.

• **Pascal constants**

• **Pascal expressions**

• **Pascal operators**

• **Pascal strings**

• **Pascal variables**

• **Program Execution**

The Run Menu command has several options for how to execute your program. Since you use these commands frequently, most are available on function keys:

VIDE

F4	Go to cursor	Alt-F7	Instruction trace
F7	Trace into		Animate
F8	Step over	Alt-F8	Until return
F9	Run	Alt-F9	Execute to
Alt-F4	Back trace		

- **Prompts**

Prompts indicate you must supply additional information to process a command. A prompt dialog has an entry field in which you can input text; you can scroll to accommodate long input lines. The dialog title describes what you must enter.

- **Scoping**

Normally, Turbo Debugger and Turbo Profiler look for a symbol in an expression in the same way that the compiler for the current language would search. For example in C, first it looks in the current function, then in the current module for a static (local) symbol, then for a global symbol. If it doesn't find the symbol using these techniques, it finally searches through all the other modules to try to find a static symbol that matches. This lets you reference static functions in other modules without having to mention the module name explicitly. Turbo Debugger uses a similar approach with the other languages.

If you want to force Turbo Debugger or Profiler to look elsewhere for a symbol, you can exert total control over where to look for a symbol name. You can specify a module, a file within a module, and/or a function to search. Where the debugger looks for a variable is known as the "scope" of that variable.

Normally you use pound signs (#) to separate the components of the scope. If it is not ambiguous in the current language, you can use dots (.) instead of #, and also omit the initial pound sign or dot.

The following syntax describes scope overriding, with brackets [] indicating optional items:

```
[#module [#filename]] #linenum [#variablename]
or
[#module [#filename]] [#funcname] #variablename
```

If you do not specify a module or file, the current module and file is assumed. For example, #123 refers to line 123 in the current module.

Scope examples:

```
#module1#myvar    "myvar" accessible from module
                  "module1"

#myfunc#var1     "var1" accessible from routine
                  "myfunc"

module2#99       Line 99 in module "module2"

m1#fil2#fun1     "fun1" accessible from file
                  "fil2" included in module "m1"
```

- **Screen updating**

- **Speed Menus**

Each pane has a speed menu for commands specific to that pane. To access the speed menu, press

Alt-F10. To invoke an item in a speed menu quickly, hold down Ctrl and press the hot key letter of the menu item. You can use the TDINST customization utility to set or cancel these control-key hot keys.

- **Stack**

There are two ways to view the stack:

- ◆ A list of active functions and the arguments they are called with. The active function calling list is displayed in a Stack window.
- ◆ A raw hex dump The hex dump of the stack contents is shown in the Stack pane of a CPU window.

- **Static variables**

Static variables are those defined as being accessible only in the module in which they are defined. The C static keyword is used as part of the declaration for the variable. You can view the static variables in a module by creating a Variables window. The static variables appear in the bottom pane of that window.

- **Status info**

A number of messages can appear either in the information box displayed with the File/Get Info command or when control returns to Turbo Debugger after executing part of your program. See TD Help for full info.

- **Technical notes**

- **Tracepoints**

You use breakpoints to specify something you want done when your program has run to a specific source line number or address.

You can set a breakpoint to stop your program at any source line or address. Position the cursor on the desired line and press F2. Press F2 again to clear the breakpoint.

You can also set a breakpoint by clicking the mouse in one of the first two columns of the line that you want to set the breakpoint on.

Turbo Debugger's breakpoints perform all the same jobs as the breakpoints, watchpoints, and tracepoints of other debuggers.

A breakpoint can be instructed to perform one of the following actions:

Break	Stop your program
Log	Log the value of variables
Execute	Execute an expression

You can set a breakpoint to execute on one of the following conditions:

Always	Every time it is encountered
Expression true	Only when an expression is true
Changed memory	Only when a memory area changes
Hardware	When a hardware breakpoint occurs

Breakpoints can also be qualified by setting a Pass Count that specifies how many times the breakpoint must be passed over before being activated.

VIDE

A breakpoint can also be set at a global address, which means that it occurs on every source line or instruction address. This allows you to watch the value of a variable as each line is executed, or to stop when a variable or area of memory changes value.

- **Variable scoping**

- **Viewers**

The View menu commands open windows that "view" part of the program you are debugging. Use the Another option in this menu to open another instance of a Dump, File, or Module window that you already have displayed on the screen.

- **Watchpoints**

See Tracepoints.

- **Windows**

You create windows with the View menu commands. The active window has a double line border and a highlighted title. Each window is divided into one or more panes. Each pane has a Speed Menu for commands specific to that pane.

Other sites with help for BCC 5.5

This section will list other non–Borland web sites that have additional information about Borland's free command line tools.

- [Helmut Pharo's Site](#)

This site has a pretty nice document about getting and setting up BCC 5.5. That information isn't much different than what is here, but the site also has a nice tutorial section for building a simple Window's GUI app with BCC 5.5, and some other nice tutorial information. Highly recommended.

Help Improve VIDE for BCC [top](#)

Please note that VIDE will remain primarily focused on the GNU MinGW gcc compiler. However, I do plan to continue support for the free Borland BCC 5.5. Remember that VIDE is GPLed, so the code is available for modification.

First, if I've gotten some default behavior wrong, please suggest a reasonable alternative. But remember, I chose the defaults here mainly to simplify typical, simple applications. Advanced users are expected to edit the Project file or makefile to get more options.

If you don't like the layout of this document (like tables might be better), chip in and fix it. I'll fold it back into the standard distribution.

Disclaimer [top](#)

First, the Borland C++ 5.5 compiler is not the main compiler supported by VIDE (gcc is). However, VIDE has supported BCC long enough now that it is very stable. If you have any problems using VIDE with BCC, please report them back to me.

VIDE

This information was assembled for publicly available sources and is intended merely to help use VIDE with Borland BCC32. There is no guarantee of its accuracy, although it seems to be correct, but may be incomplete.

No Warranty [top](#)

This program is provided on an "as is" basis, without warranty of any kind. The entire risk as to the quality and performance of the program is borne by you.

VIDE Reference Manual

Copyright © 1999–2000, Bruce E. Wampler

All rights reserved.

Bruce E. Wampler, Ph.D.

bruce@objectcentral.com

www.objectcentral.com