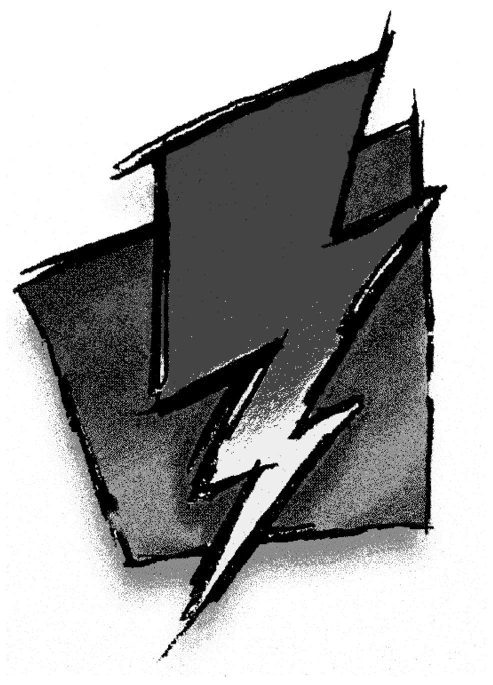
Watcom C/C++

Getting Started



Edition 11.0c

Notice of Copyright

BA0614

Table of Contents

Table of Contents

Table of Contents

1 Introduction to Watcom C/C++

Welcome to the Watcom C/C++ 11.0 development system. Version 11.0 of Watcom C/C++ is a professional, optimizing, multi-platform C and C++ compiler with a comprehensive suite of development tools for developing and debugging both 16-bit and 32-bit applications for DOS, extended DOS, Novell NLMs, 16-bit OS/2 1.x, 32-bit OS/2, Windows 3.x, Windows 95, Win32s, and Windows NT (Win32).

You should read the entire contents of this booklet, as it contains information on new programs and modifications that have been made since the previous release.

Special NOTE to users of previous versions! See the section entitled "Release Notes for Watcom C/C++ 11.0" on page 63 **to determine if you need to recompile your application.**

1.1 What is in version 11.0 of Watcom C/C++?

Version 11.0 incorporates the features professional developers have been demanding:

Open, Multi-target Integrated Development Environment

The IDE allows you to easily edit, compile, link, debug and build applications for 16-bit systems like DOS, OS/2 1.x, and Windows 3.x and 32-bit systems like extended DOS, Novell NLMs, OS/2, Windows 3.x (Win32s), Windows 95, and Windows NT. Projects can be made up of multiple targets which permit a project to include EXEs and DLLs. The IDE produces makefiles for the project which can be viewed and edited with a text editor. The IDE is hosted under Windows 3.x, Windows 95, Windows NT, and 32-bit OS/2.

The Widest Range of Intel x86 Platforms

Host Platforms

- DOS (command line)
- 32-bit OS/2 (IDE and command line)
- Windows 3.x (IDE)
- Windows 95 (IDE and command line)
- Windows NT (IDE and command line)

16-bit Target Platforms

- DOS
- Windows 3.x
- OS/2 1.x

32-bit Target Platforms

- Extended DOS
- Windows 3.x using our 32-bit extender technology
- Win32s
- Windows 95
- Windows NT
- 32-bit OS/2
- Novell NLMs
- AutoCAD ADS

Cross-Platform Development Tools
The core tools in the package permit cross-platform development that allows developers to exploit the advanced features of today's popular 32-bit operating systems, including Windows 95, Windows NT, and OS/2. Cross-platform support allows you to develop on a host development environment for execution on a different target system.

Multi-Platform Debugger
The new debugger advances developer productivity. New features include redesigned interface, ability to set breakpoints on nested function calls, improved C++ and DLL debugging, reverse execution, and configurable interface. Graphical versions of the debugger are available under Windows 3.x, Windows 95, Windows NT, and 32-bit OS/2. Character versions of the debugger are available under DOS, Windows 3.x, Windows NT, and 32-bit OS/2. For VIDEO fans, we have kept the command line compatibility from the original debugger.

Class Browser
The Browser lets you visually navigate the object hierarchies, functions, variable types, and constants of your C/C++ application.

Performance Analysis
The Watcom Execution Sampler and Watcom Execution Profiler are performance analysis tools that locate heavily used sections of code so that you may focus your efforts on these areas and improve your application's performance.

Editor The Watcom Editor is a context sensitive source editor, integrated into the Windows 3.x, Windows 95 and Windows NT version of the IDE.

Visual Programmer

Blue Sky's Visual Programmer is incorporated in the package. Visual Programmer is powerful visual design environment for Windows 95 and Windows NT that helps you build sophisticated Windows applications. Visual Programmer provides the familiar elements of the Windows user interface, such as pull-down menus, toolbars, dialog boxes, controls, and other elements as ready-made objects that you can incorporate into your application instantly. In the visual design environment, you paint your user interface directly on the screen and attach functionality to the interface components by visually connecting them on the screen, without writing a single line of code.

Graphical Development Tools

Watcom C/C++ includes a suite of graphical development tools to aid development of Windows 3.x, Windows 95 and Windows NT applications. The development tools include:

Resource Editors	Enable you to create resources for your 16-bit and 32-bit Windows applications. For 32-bit OS/2 PM development, Watcom C/C++ includes IBM's resource editors. These tools have been seamlessly integrated into the IDE. The resource compiler allows you to incorporate these resources into your application.
Resource Compiler	Produces a compiled resource file from a source file.
Zoom	Magnifies selected sections of your screen.
Heap Walker	Displays memory usage for testing and debugging purposes.
Spy	Monitors messages passed between your application and Windows.
DDESpy	Monitors all DDE activity occurring in the system.
Dr. Watcom	Enables you to debug your program by examining both the program and the system after an exception occurs; monitors native applications running under Windows 3.x, Windows 95 or Windows NT.

Assembler

An assembler is now included in the package. It is compatible with a subset of the Microsoft assembler.

MFC Support
Watcom C/C++ allows you to recompile existing MFC applications for use under Windows 3.x, Windows NT and Win32s. MFC 4.1 is supported for 32-bit applications under Windows NT, Windows 95 and Win32s. MFC 2.52b is supported for 16-bit applications under Windows 3.x.

Licensed components of OS/2 2.1 Toolkit
Includes the full OS/2 2.1 Presentation manager and character mode APIs, on-line help and example programs.

Licensed components of Windows 3.1 Toolkit
Includes the full Windows 3.1 API libraries and on-line help.

Licensed components of Windows NT Toolkit
Includes the full Windows NT API libraries and on-line help.

Licensed components of Novell NLM SDK 4.0
Includes all header and import files needed to create an NLM.

Licensed components of the SOMobject's Developer Toolkit for OS/2
Watcom C/C++ supports native binding for both C and C++ applications.

C++ Class Libraries
Watcom C/C++ includes container and stream class libraries.

Royalty-free 32-bit DOS Extender
Watcom C/C++ includes the DOS/4GW 32-bit DOS extender by Tenberry Software with royalty-free run-time and virtual memory support up to 32MB.

Support for wide range of DOS Extenders
Watcom C/C++ allows you to develop and debug applications based on the following DOS extender technology: Tenberry Software's DOS/4G and Phar Lap's TNT DOS Extender. You can also develop applications using FlashTek's DOS Extender but, currently, there is no support for debugging these applications.

Sample programs and applications
Watcom C/C++ includes a large set of sample applications to demonstrate the new integrated development environment.

1.2 Technical Support and Services

We are committed to ensuring that our products perform as they were designed. Although a significant amount of testing has gone into this product, you may encounter errors in the software or documentation. Our technical support offers a variety of services to help you work around any problems you may encounter. Watcom C/C++ 11.0 includes 60-days of free installation assistance. Installation assistance will include installing the product and compiling and running the samples that are included with the product for your specific environment. Subsequent to this 60-day period, technical assistance is available through our fee-based support programs. Please contact us for more information.

In order to keep informed about product updates and announcements, we suggest that you send in your registration card. Product registration cards are included in the package and can be mailed or faxed to us.

Resources at Your Fingertips

Watcom C/C++ contains many resources to help you find answers to your questions. The documentation is the first place to start. With each release of the product, we update the manuals to answer the most frequently asked questions. Most of this information is also accessible through on-line help.

The "README" file in the main product directory contains up-to-date information that recently became available.

Answers to frequently asked questions are available on CompuServe, Powersoft's World Wide Web server, Powersoft's FTP site, Powersoft's bulletin board, or the Powersoft's Faxline fax back system. These services are available 24 hours a day. See the Powersoft ***Customer Services Reference Guide*** for more information on *Automated Technical Support (ATS).*

Contacting Technical Support

Our technical support is available to help resolve technical defects in the software. The following are ways to contact technical support.

Telephone The telephone number for the Technical Support office nearest to you is listed in the ***Customer Services Reference Guide.***

Fax The telephone number of the facsimile machine for the Technical Support office nearest to you is listed in the ***Customer Services Reference Guide.***

World Wide Web You can also send bug reports or enhancement requests to Technical Support using the Powersoft World Wide Web server. Electronic forms are available for on-line completion.

FTP You can obtain a copies of the bug report or enhancement request forms from the Powersoft FTP site.

```
ftp://ftp.powersoft.com/pub/watcom/general/wbugrep.zip
ftp://ftp.powersoft.com/pub/watcom/general/wenhreq.zip
```

The bug report form (WBUGREP.TXT) is available on Compuserve, the Powersoft BBS, and Faxline (document #1014). The enhancement request form (WENHREQ.TXT) is available on Compuserve, the Powersoft BBS, and Faxline (document #1015).

Information Technical Support Will Need to Help You

Your registration number will be required when you contact Technical Support.

The more information you can provide to your technical support representative, the faster they can help you solve your problem. A detailed description of the problem, short sample program, and a summary of steps to duplicate the problem (including compiler and linker options) are essential. Concise problem reports allow technical support to quickly pinpoint the problem and offer a resolution. Here is a list of information that will help technical support solve the problem:

Contact information
We would like your name, as well as telephone and fax numbers where you can be reached during the day.

Product information
Please tell us the product name, version number, patch level, and software registration number.

Hardware configuration
Please tell us what type of processor you are using (e.g., 33MHz 486SX), how much memory is present, what kind of graphics adapter you are using, and how much memory it has.

Software configuration
Please tell us what operating system and version you are using.

Output from the TECHINFO utility
Please provide the output from the Techinfo utility. It lists current patch levels, environment variable settings, and the contents of your AUTOEXEC.BAT and CONFIG.SYS files.

Concise problem report with short sample program
Please provide a complete description of the problem and the steps to reproduce it. A small, self-contained program example with compile and link options is ideal.

An electronic form of communication is often preferable for reporting technical problems. It provides an easy and efficient way to receive sample code and complete problem details.

Making Fixes Available to Developers: The Patch Utility

We are dedicated to fixing problems in our products. We developed the patch utility to enable timely responses to bugs in the software. Patches are frequently made available and are designed to address problems with the software. When you call technical support one of the first questions asked is your current patch level.

How do I determine the current patch level?
Here are two easy ways for you to determine your current patch level:

1. At the command line, type the command "techinfo". Among other information, it displays the patch level of the various components in the package.

2. If you are using the command line tools, both the C and C++ compiler display a banner at startup that echo the version number with the patch level.

How can I get the patches?
Patches are made available on CompuServe, our World Wide Web server, our FTP site, and our bulletin board. They are available 24 hours a day. Patches are also available on the current release CD-ROM.

How do I apply the patches?
Once you have downloaded the patches, issue the `pkunzip` command to unzip the patch files. A "README" file provides information on the patch.

How can I ensure the patch applied correctly?
Running the Techinfo utility indicates the current patch level of the tools. Ensure that the patch indication for the files is the one you just applied.

1.3 Third-party Support

A large number of third party software packages support the use of the Watcom C/C++. We have included a document on the CD-ROM which describes some of these packages. The information on these software packages was provided by the developers of these packages. The document takes the form of a help file, of which versions are provided for DOS, Windows 3.x, Windows 95, Windows NT, and OS/2.

1.4 Suggested Reading

There are a number of good books and references that can help you answer your questions. Following is a list of some of the books and documents we feel might be helpful. This is by no means an exhaustive list. Contact your local bookstore for additional information.

C Programmers

The C Programming Language, 2nd Edition
Brian W. Kernighan and Dennis M.Ritchie; Prentice Hall, 1988.

C DiskTutor
L. John Ribar; Osborne McGraw-Hill, 1992.

C++ Programmers

C++ Primer, 2nd Edition
Stanley B. Lippman; Addison-Wesley Publishing Company, 1991.

Teach Yourself C++ in 21 Days
Jesse Liberty; Sams Publishing, 1994.

DOS Developers

PC Interrupts, Second Edition
Ralf Brown and Jim Kyle; Addison-Wesley Publishing Company, 1994.

Relocatable Object Module Format Specification, V1.1
The Intel OMF specification can be obtained from the Intel ftp site. Here is the URL.

```
ftp://ftp.intel.com/pub/tis/omf11g.zip
```

This ZIP file contains a Postscript version of the Intel OMF V1.1 specification.

Extended DOS Developers

Extending DOS—A Programmer's Guide to Protected-Mode DOS, 2nd Edition
Ray Duncan, et al; Addison-Wesley Publishing Company, 1992.

DOS Protected-Mode Interface (DPMI) Specification
The DPMI 1.0 specification can be obtained from the Intel ftp site. Here is the URL.

```
ftp://ftp.intel.com/pub/IAL/software_specs/dpmiv1.zip
```

This ZIP file contains a Postscript version of the DPMI 1.0 specification.

Windows 3.x Developers

Microsoft Windows Programmer's Reference
Microsoft Corporation; Microsoft Press, 1990.

Programming Windows 3.1, Third Edition
Charles Petzold; Microsoft Press, 1992.

Windows Programming Primer Plus
Jim Conger; Waite Group Press, 1992.

Windows NT Developers

Advanced Windows NT
Jeffrey Richter; Microsoft Press. 1994.

Inside Windows NT
Helen Custer; Microsoft Press. 1993.

Microsoft Win32 Programmer's Reference, Volume One
Microsoft Corporation; Microsoft Press, 1993.

OS/2 Developers

The Design of OS/2
H.M. Deitel and M.S. Kogan; Addison-Wesley Publishing Company, 1992.

OS/2 Warp Unleashed, Deluxe Edition
David Moskowitz and David Kerr, et al; Sams Publishing, 1995.

OS/2 Technical Library.
To order the Technical Library, call one of the following numbers.

```
In Canada:             1-800-465-1234
In the United States: 1-800-426-7282 (OS/2 2.x)
                       1-800-879-2755 (OS/2 Warp)
```

You can also order copies of these books from an IBM authorized dealer or IBM representative.

Virtual Device Driver Developers

Writing Windows Virtual Device Drivers
David Thielen and Bryan Woodruff; Addison-Wesley Publishing Company, 1994.

2 Installation

The package contains the following components:

- *Watcom C/C++ CD-ROM*
- *This manual*
- *Registration Card*
- *Envelope containing important information*
- *Customer Services Reference Guide*

2.1 Send in Your Registration Card

You should fill out and mail your registration card as soon as possible to ensure you are informed of future upgrades and other special offers to registered users. Your registration number is printed on the registration card. **Retain the other portion of the registration card for your records. Your registration number is printed on it. You will need this number when you contact Technical Support.**

2.2 Hardware and Software Requirements

Watcom C/C++ requires the following minimum configuration:

- IBM PC compatible

- An 80386 or higher processor

- 8 MB of memory

- 441 MB of disk space.

Note that the disk space requirement varies with the disk cluster size. The larger the cluster size, the greater the disk requirement. If your hard disk has 8K clusters (257MB to 512MB), a complete installation will require 441 MB megabytes of disk space. If your hard disk has 16K clusters (513MB to 1GB), a complete installation will require 514 MB megabytes. A selective install will require considerably less space.

- A CD-ROM disk drive

In addition to the above requirements, you need one of the following operating systems:

- DOS version 5.0 or higher
- Microsoft Windows version 3.1 running in enhanced mode
- Microsoft Windows 95
- Microsoft Windows NT version 3.1 or higher
- IBM OS/2 2.1 or higher

You must install the software from a CD-ROM drive. Unlike earlier releases, it is now no longer possible to copy the installation files to diskettes and install the software from your diskette drive.

2.3 The README File

Before you install Watcom C/C++, you should read the contents of the "README" file which is stored in the root directory of the CD-ROM. It contains valuable, up-to-date information concerning this product.

2.4 Installing Watcom C/C++

The installation program in this version has been completely redesigned with several new "smart" features. If you have installed a previous version of Watcom C/C++ then you should install Watcom C/C++ 11.0 into the same path (except for the reason described in the following paragraph). It will examine a previous installation to determine what features were previously installed. It will use this information to establish default settings for the installation that you are about to attempt. Of course, you can add or remove features as you progress through the installation steps.

If you are installing only one of the Watcom C/C++ or Watcom FORTRAN 77 products and you have an older version of the other product, we do NOT recommend that you install the new product into the same directory as the old product. The Watcom C/C++ and Watcom FORTRAN 77 products are compatible at the same version number. However, the Watcom C/C++ and Watcom FORTRAN 77 products are usually NOT compatible across different version numbers (e.g., version 9.5 of Watcom FORTRAN 77 is not compatible with version 10.5 of Watcom C/C++ and vice versa). If this is the case, care must be exercised when switching between use of the two products. Environment variables such as **PATH** and **WATCOM** must be modified and/or corrected. System files such as `CONFIG.SYS` and `SYSTEM.INI` must be modified and/or corrected.

If you are installing both Watcom C/C++ 11.0 and Watcom FORTRAN 77 11.0, we recommend that you install both products under the same directory. This will eliminate duplication of files and, as a result, reduce the total required disk space. The two products share the use of certain environment variables which point to the installation directory. If separate installation directories are used, problems will arise.

When you install Watcom C/C++ and Watcom FORTRAN 77 in the same directory, you should not deselect any options when running the second installation; otherwise the second product's install may remove files that were installed (and are required) by the first product's install. This isn't an issue if you only have one of Watcom C/C++ or Watcom FORTRAN 77. The problem is that Watcom C/C++ and Watcom FORTRAN 77 don't know about the installation options you have selected for each other's product.

If you wish to create a backup of your previous version, please do so before installing Watcom C/C++ 11.0.

If you decide to install Watcom C/C++ 11.0 into a different directory than the previously installed version, you will have to manually edit system files (e.g., CONFIG.SYS, AUTOEXEC.BAT, SYSTEM.INI) after the installation process is complete to remove the old version from various environment variables (e.g., PATH, DEVICE=). This is necessary since the path to the new version will appear after the path to the old version. To avoid this extra work, we recommend installing the new version into the same path as the old version.

As an example, here are a few of the environment variables and "RUN" directives that are modified/added to the OS/2 `CONFIG.SYS` file. You should make sure that all references to the older version of the software are removed.

Example:
```
    LIBPATH=...;D:\WATCOM\BINP\DLL;...
    SET PATH=...;D:\WATCOM\BINP;D:\WATCOM\BINW;...
    SET HELP=...;D:\WATCOM\BINP\HELP;...
    SET BOOKSHELF=...;D:\WATCOM\BINP\HELP;...
    SET INCLUDE=...;D:\WATCOM\H\OS2;D:\WATCOM\H;
    SET IPFC=D:\WATCOM\TOOLKT2X\IPFC
    SET WATCOM=D:\WATCOM
    SET EDPATH=D:\WATCOM\EDDAT
    RUN=D:\WATCOM\BINP\BATSERV.EXE
    RUN=D:\WATCOM\BINP\NMPBIND.EXE
```

You may wish to run Watcom C/C++ under more than one operating system on the same personal computer. For every operating system that you use, simply start up the operating system and run the corresponding install procedure.

If you run the Windows 3.x installation procedure, you do not need to run the DOS installation procedure also.

If you plan to use Win-OS/2 as a development platform under OS/2, you must run the Windows 3.1 install program (selecting Windows 3.1 host support).

Place the CD-ROM disk in your CD-ROM drive. Select one of the following procedures depending on the host operating system that you are currently running. Below, substitute the CD-ROM drive specification for "x:".

DOS Enter the following command:

```
x:\setup
```

Windows 3.x Start Windows 3.x and choose Run from the File menu of the Program Manager. Enter the following command:

```
x:\setup
```

Windows 95 Choose Run from the Start menu and enter the following command:

```
x:\setup
```

Windows NT Log on to an account that is a member of the "Administrator" group so that you have sufficient rights to modify the system environment. Choose Run from the File menu of the Program Manager. Enter the following command:

```
x:\setup
```

OS/2 Start an OS/2 session and enter the following command:

```
x:\install
```

2.5 Incremental Installation

You may wish to install Watcom C/C++, and subsequently install features that you omitted in the first install. You can also remove features that you no longer wish to have installed. You can achieve this as follows:

1. Start the installation program.
2. Select any new features that you wish to install.
3. Deselect any features that you wish to remove.
4. Re-run the installation program for each host operating system that you use.

2.6 System Configuration File Modifications

The install program makes changes to your operating system startup files to allow Watcom C/C++ to run. We strongly recommend that you allow the install program to modify your system configuration files for you, but you may do it by hand. The changes required may be found in any of the following files which have been placed in the root of the installation directory:

CONFIG.NEW	Changes required for `CONFIG.SYS` (DOS, Windows, Windows 95, OS/2)
AUTOEXEC.NEW	Changes required for `AUTOEXEC.BAT` (DOS, Windows, Windows 95, OS/2)
CHANGES.ENV	Changes required for the Windows NT environment

2.7 Installation Notes for Windows 3.x

1. When you use the Integrated Development Environment under Windows 3.x, it is important that the IDE's batch server program be able to run in the background. Therefore, make sure that the "Exclusive in Foreground" checkbox is NOT checked in the "Scheduling" options of "386 Enhanced" in the "Control Panel".

2. When you use the Integrated Development Environment under Windows 3.x, the line

   ```
   OverlappedIO=ON
   ```

   in your "SYSTEM.INI" file can cause problems. This controls (disables) the queuing of DiskIO and makes some changes between DOS box timings to allow some processes to finish.

3. When you use the Integrated Development Environment under Windows 3.x, it is important that the line

   ```
   NoEMMDriver=ON
   ```

   not appear in your "SYSTEM.INI" file. It will prevent a link from succeeding in the IDE..

4. When you use the Integrated Development Environment under Windows 3.x on the NEC PC-9800 series, it is important that the line

```
InDOSPolling=TRUE
```

not appear in your "SYSTEM.INI" file. It will prevent a make from succeeding in the IDE.

5. Central Point Software's anti-virus programs (VDEFEND, VSAFE, VWATCH) conflict with the Integrated Development Environment under Windows 3.x.

6. The Program Information File "BATCHBOX.PIF" is used by the Integrated Development Environment (IDE) to start up a background batch server for compiling, linking, etc. The PIF references "COMMAND.COM". If you are using a substitute for "COMMAND.COM" such as "4DOS.COM" then you must modify the PIF accordingly using a PIF editor.

2.8 Installation Notes for OS/2

1. The Integrated Development Environment (IDE) uses the IBM OS/2 Enhanced System Editor (EPM) for editing text files. You must ensure that EPM is installed in your OS/2 system if you are planning to use the IDE. You can selectively install the Enhanced Editor by running the OS/2 Setup and Installation program (Selective Install) and choosing "Enhanced Editor" from the "Tools and Games" detail page.

2. The Integrated Development Environment (IDE) requires that the `BATSERV.EXE` program be started during OS/2 initialization. If you plan to use the IDE, do not remove the "RUN=" line for this file from your `CONFIG.SYS` file.

3. On some systems with limited memory that use the UNDELETE feature of OS/2, compile times may be slow because OS/2 is saving copies of compiler temporary files. You may start the BATSERV process using the OS/2 `STARTUP.CMD` file with **DELDIR** turned off as illustrated below.

```
SET OLD_DEL_DIR=%DELDIR%
SET DELDIR=
DETACH C:\WATCOM\BINP\BATSERV.EXE
SET DELDIR=%OLD_DEL_DIR%
SET OLD_DEL_DIR=
```

4. If you plan to use the Named Pipe Remote Debugging support of the Watcom Debugger then the `NMPSERV.EXE.` program must be started during OS/2 initialization. If you plan to use this feature, do not remove the "RUN=" line for this file from your `CONFIG.SYS` file.

2.9 Installation Notes for Win32s

There is a directory called WIN32S in the SDK directory of the CD-ROM. It contains the entire Win32s directory structure. The Watcom C/C++ install program will install Win32s onto your computer system, but it will not install the debug version of Win32s. If you want to use the debug version of Win32s, type the commands:

```
X:
cd \sdk\win32s
install D:
```

where "X" is the drive letter of your CD-ROM drive, and "D" is the drive you want to install Win32s on. Once you have done this, you can use `SWITCH.BAT` in the `WIN32S\BIN` directory to switch between the debug and non-debug versions of Win32s.

As an alternative to installing Win32s using the Watcom C/C++ install program, you can run `X:\SDK\WIN32S\DISKS\DISK1\SETUP.EXE.`

3 Hands-on Introduction to Watcom C/C++

Let’s get started and introduce some of the new tools that are in version 11. The purpose of this chapter is to briefly test out the new graphical tools in version 11 using an existing application.

In this tutorial, we will take an existing set of C++ source files, create a project in our integrated development environment, and perform the following tasks:

- Add multiple targets
- Make a target
- Make all targets
- Execute the program
- Debug the program
- Use the Browser
- Correct errors
- Sample and profile the executable
- Save the project
- Terminate the session

3.1 Outline

Watcom’s Integrated Development Environment (IDE) manages the files and tools that a programmer uses when developing a project. This includes all the source files, include files, libraries, compiler(s), linkers, preprocessors, etc. that one uses.

The IDE has a graphical interface that makes it easy to visualize the make-up of a project. A single IDE session shows a project. If the project consists of a number of components, such as two executables and one library, these are each shown as target windows in the project window. Each target window shows the files that are needed to construct the target and is associated via its filename extension with a rule that describes the construction mechanism. For example, a filename with the extension ".EXE" may be associated with the rule for constructing 32-bit Windows executables, or a filename with the extension ".LIB" may be associated with the rule for constructing static libraries. Different projects can refer to the same target. If they do, the target is shared and can be manipulated via either project, with changes made through one affecting the other.

The IDE itself is a collection of programs that manages the various files and tools used to create the target libraries and executables. It creates makefile(s) from the information in the target descriptions and invokes Watcom Make to construct the targets themselves. A configuration file contains built-in knowledge of the Watcom compilers, editors, Profiler, and Browser, as well as all their switches.

3.2 The Watcom C/C++ Tutorial

This tutorial walks you through the creation and execution of a C/C++ program under Windows. This will give you an understanding of the basic concepts of the IDE and its components, and it will detail the steps involved in project development. The result of this tutorial is a three dimensional drawing of a kitchen which you can manipulate using either the menus or the icons on the toolbar. You can rotate and resize the drawing, as well as adjust the lighting and contrast.

To begin, start the IDE. This is done by double-clicking on the "IDE" icon in the Watcom C/C++ window.

A status field at the bottom of the IDE window indicates the function of the icon on the toolbar over which your mouse cursor is currently positioned. If the status area does not show you the function of the icons as you move the mouse cursor over them, check that no item in the menu bar is highlighted (if one is highlighted, press the Alt key).

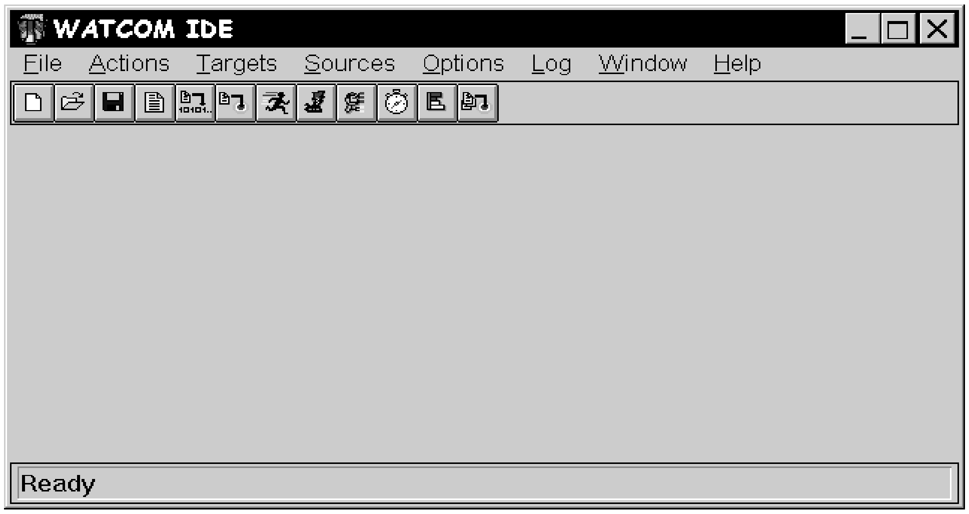


Figure 1. *The initial IDE screen*

Defining a Project

In this tutorial, you will be creating a new project called `KITCHEN.` Here are the steps required to accomplish this task.

1. Define a new project by pulling down the *File* menu and selecting the *New Project...* item. You can also define a new project by clicking on the "Create a new project" icon on the toolbar.

2. A choice of different sample project directories is available. Assuming that you installed the Watcom C/C++ software in the `\WATCOM` directory, you will find the sample project directories in the following directory:

   ```
   \WATCOM\SAMPLES\IDE
   ```

   For purposes of this tutorial, we recommend that you select one of the following project directories:

WIN	for an example of 16-bit Windows 3.x application development when using Windows 3.x under DOS as a host development system,
WIN386	for an example of 32-bit Windows 3.x application development when using Windows 3.x under DOS as a host development system,
WIN32	for an example of 32-bit Win32 application development when using Windows NT or Windows 95 as a host development system, and
OS2	for an example of 32-bit OS/2 application development when using 32-bit OS/2 as a host development system.

   Thus the `target` that we refer to below should be one of `WIN, WIN386, WIN32,` or `OS2` depending on your selection. The tutorial uses the `WIN32` example for illustrative purposes. You will find some minor variations from your selected target environment.

   When asked for a project name, you can do one of two things:

   1. enter the following pathname:

      ```
      d:[path]\SAMPLES\IDE\target\KITCHEN
      ```

      where `d:[path]` is the drive and path where you installed the Watcom software, or

2. use the file browser to select the following directory:

   ```
   d:[path]\SAMPLES\IDE\target
   ```

   and specify the filename `KITCHEN.`

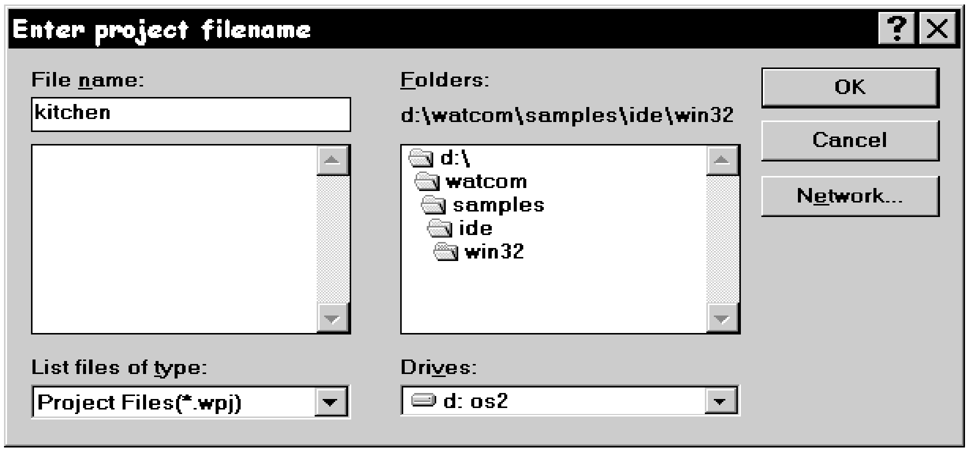


Figure 2. *Creating a new project*

Press the Enter key or click on OK (OPEN).

The project description will be stored in this file and the IDE will set the current working directory to the specified path during your session.

3. You will be prompted for a target name. Since we will be attaching pre-defined targets, just click the *Browse* button when prompted for the target name. Select the "draw" target file (it will be one of `DRAW16.TGT`, `DRAW.TGT`, `DRAW32.TGT`, `DRAWOS2.TGT` depending on your selection of target).

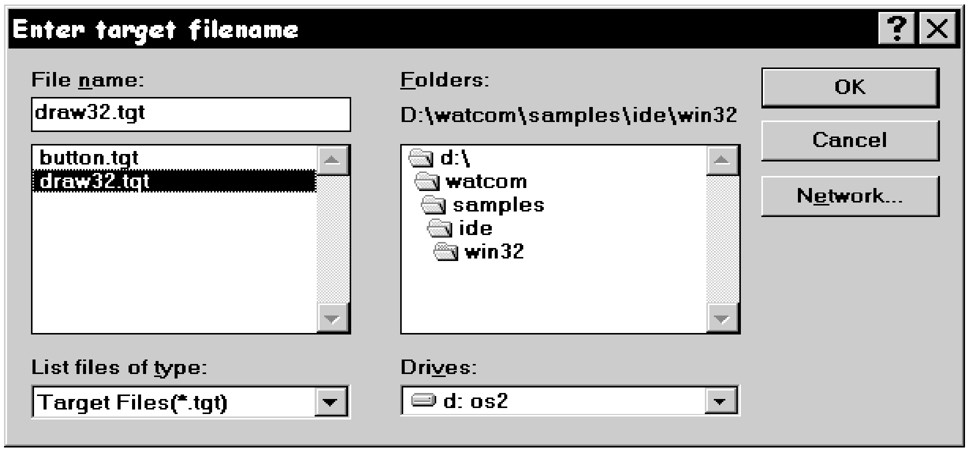
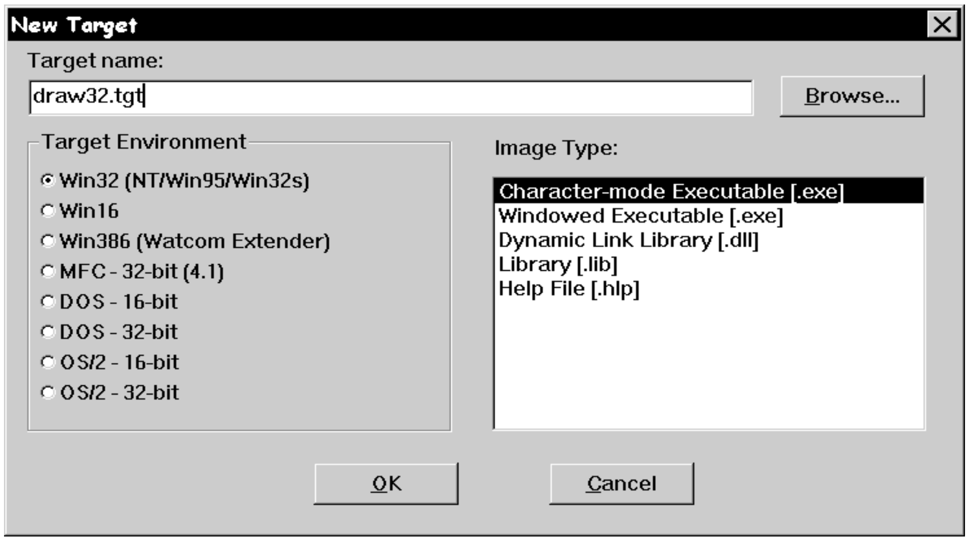


Figure 3. *Attaching existing targets*

Press the Enter key or click on OK (OPEN).

4. You can ignore the settings displayed for *Target Environment* and *Image Type* since the target definition already exists (we created it for you). The settings are important when you a defining a new target (i.e., one that was not predefined).


New Target

Target name: draw32.tgt

Target Environment:
- [✓ Win32 (NT/Win95/Win32s)]
- Win16
- Win386 (Watcom Extender)
- MFC - 32-bit (4.1)
- DOS - 16-bit
- DOS - 32-bit
- OS/2 - 16-bit
- OS/2 - 32-bit

Image Type:
- [✓ Character-mode Executable [.exe]]
- Windowed Executable [.exe]
- Dynamic Link Library [.dll]
- Library [.lib]
- Help File [.hlp]

Figure 4. *Selecting a target type*

Press the Enter key or click on OK.

A target window is created in the project window for the "draw" target. This window contains all of the files associated with the target. You can click on any of the "Folder" icons to hide or un-hide all files with a particular extension. For example, you may wish to un-hide all the files with a `.BMP` extension by clicking on the folder icon associated with bitmap files.

Adding Multiple Targets

Watcom’s IDE allows you to have multiple targets in any particular project. Note that targets can be used by multiple projects. To add a new target to the project, do the following.

1. Pull down the *Targets* menu and select the *New Target...* item.

2. Enter `BUTTON.TGT` as the target name for the new target to be added to the project. Do not forget to include the `.TGT` extension. It is required when selecting a pre-existing target.

3. You can ignore the settings displayed for *Target Environment* and *Image Type* since the target definition already exists (we created it for you). The settings are important when you a defining a new target (i.e., one that was not predefined).

4. Press the Enter key or click on OK.

A target window is created in the project window for the `button` target.

Making a Target

Watcom’s IDE will automatically generate the sequence of steps required to build or "make" each of the targets in a project. Note that the targets in a project can be made individually or collectively. To make the `button.lib` target, do the following.

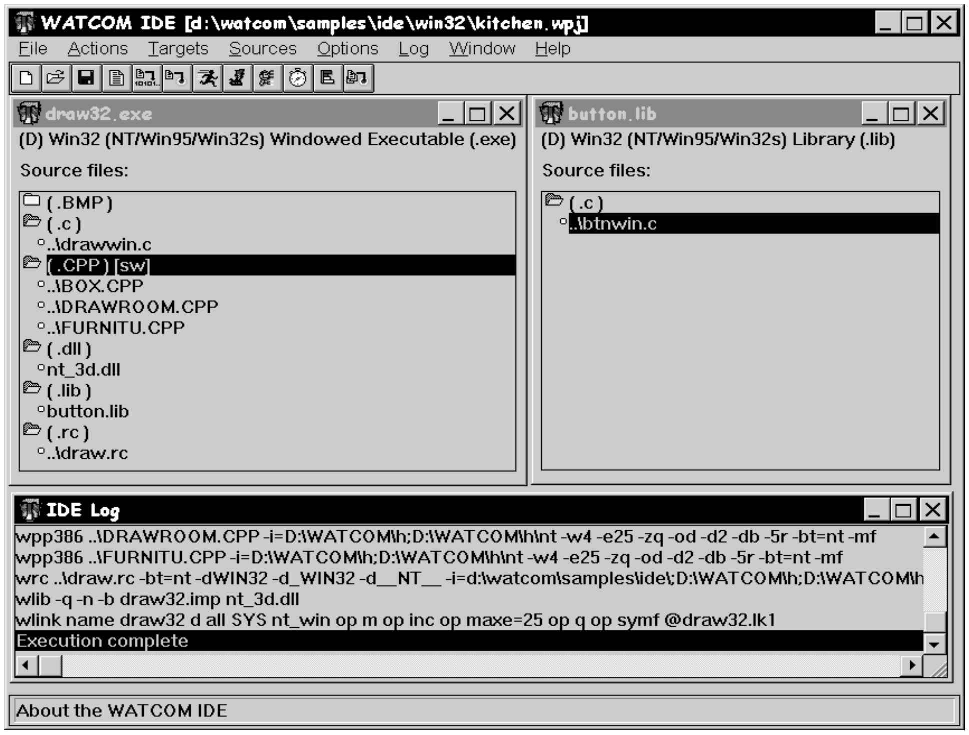
1. Click on the window of the target you wish to make. In this case, click on the `button.lib` target window.

2. Pull down the *Targets* menu and select the *Make* item (you can also do this by clicking on the "Make the current target" icon on the toolbar, or by right-clicking on the target and selecting the *Make* item from the pop-up menu).

The IDE will now construct a makefile based on its knowledge of the target and construction rules, and then invoke the make utility to create the target, in this case `button.lib.` The output of this procedure is displayed in the Log window.

Making All Targets

Click the "Make all targets in the project" icon on the toolbar to make all of the targets in the current project. If one target depends on another target, the latter target will be made first. In this tutorial `BUTTON.LIB` will be made first (there will be nothing to do since it was made previously) and then `DRAW???.EXE,` since `BUTTON.LIB` is in the list of files associated with `DRAW???.EXE.` In each case, the IDE constructs makefiles based on its knowledge of the target and construction rules. The output of this procedure is displayed in the Log window.



```
wpp386 ..\DRAWROOM.CPP -i=D:\WATCOM\h;D:\WATCOM\h\nt -w4 -e25 -zq -od -d2 -db -5r -bt=nt -mf
wpp386 ..\FURNITU.CPP -i=D:\WATCOM\h;D:\WATCOM\h\nt -w4 -e25 -zq -od -d2 -db -5r -bt=nt -mf
wrc ..\draw.rc -bt=nt -dWIN32 -d_WIN32 -d__NT__ -i=d:\watcom\samples\ide\;D:\WATCOM\h;D:\WATCOM\h
wlib -q -n -b draw32.imp nt_3d.dll
wlink name draw32 d all SYS nt_win op m op inc op maxe=25 op q op symf @draw32.lk1
Execution complete
```

Figure 5. *Making one or more targets*

Executing the Program

The project should have built without errors, so now you are ready to execute the program you have developed.

Click on the `DRAW???.EXE` target window and simply click on the "Run the current target" icon on the toolbar. You can also do this by selecting *Run* from the *Targets* menu, or right-clicking anywhere on the target window outside of the inner frame and selecting *Run* from the pop-up menu (right-clicking is context sensitive and the pop-up menu that results will vary depending on the area of the window in which you right-click).

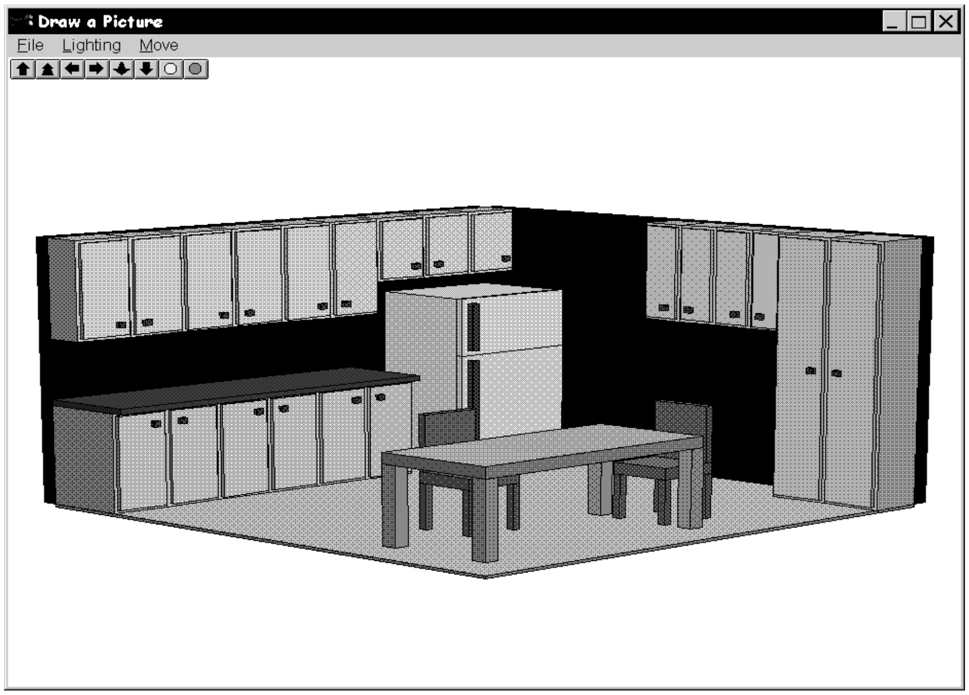
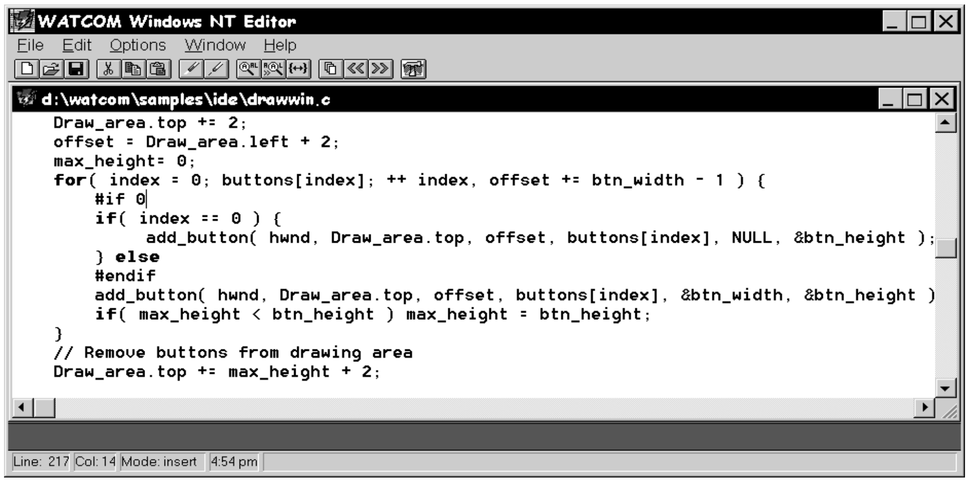


Figure 6. *The kitchen demo*

The demo you have created is a simple three dimensional drawing of a kitchen. By using either the icons on the toolbar or the menus you can rotate the picture left, right, up, and down, make the picture brighter or dimmer, move the picture closer or farther away, and increase or decrease the amount of contrast (this latter feature is found in the "Lighting" menu). Choose *Exit* from the *File* menu to exit the demo program when you are finished.

Smart Editing

The IDE recognizes the type of file you wish to edit, and invokes the appropriate editor for the task. To edit a file, you either double-click on it or select it and click the "Edit" icon on the toolbar. Files with a `.C, .CPP, .H, .HPP, .FOR, .ASM` or `.RC` extension are edited with a text editor; files with a `.BMP, .ICO,` or `.CUR` extension are edited with the Image Editor; files with a `.DLG` or `.RES` extension are edited with the Resource Editor.


WATCOM Windows NT Editor
File Edit Options Window Help
d:\watcom\samples\ide\drawwin.c

```
    Draw_area.top += 2;
    offset = Draw_area.left + 2;
    max_height= 0;
    for( index = 0; buttons[index]; ++ index, offset += btn_width - 1 ) {
        #if 0
        if( index == 0 ) {
            add_button( hwnd, Draw_area.top, offset, buttons[index], NULL, &btn_height );
        } else
        #endif
        add_button( hwnd, Draw_area.top, offset, buttons[index], &btn_width, &btn_height )
        if( max_height < btn_height ) max_height = btn_height;
    }
    // Remove buttons from drawing area
    Draw_area.top += max_height + 2;
```

Figure 7. *The Watcom Editor for Windows*

Now we will edit one of the source files and introduce an error into the application.

1. Double-click on the "draw" source file (i.e., `DRAW???.C`) to load the source file into the editor.

2. Scroll down to line 217 using the keyboard or mouse. You can also pull down the *Edit* menu, select *Goto Line...*, and enter 217. The Watcom Editor makes full use of colors and fonts to achieve syntax highlighting. File templates for C, C++, and FORTRAN files are defined to assist you in distinguishing the components of your code.

3. Pull down the *Options* menu and select the *Colors* item.

4. Click on a color from the palette, drag it to the word `if` on line 218, and release it. All keywords are now displayed in the chosen color. Drag a different color to a comment line (line 225) and all comments will display in that color. Similarly, you can select the Fonts item from the Options menu, select a font style and size, and drag it to your source file. Close the Fonts and Colors dialog by double-clicking in the upper left hand corner.

5. You can now save this color and font configuration for all `.CPP` files by pulling down the *Options* menu, selecting the *General...*, item and clicking next to *Save configuration on exit* in the "Features" box. Press Enter or click on OK.

6. Now, to introduce an error into the application, replace the line `#if 0` with `#if 1.`

7. Save your changes by clicking on the "Write the current file to disk" icon or select *Save* from the *File* menu.

8. Return to the IDE (by clicking on it if it is visible on your screen, or by using Alt-Tab), re-make your project, and run it. A fault occurs in your application, so the next step is to track down the problem using the Watcom Debugger.

Debugging the Program

To debug a program it must first be compiled to include debugging information. The IDE maintains two sets of switches for each target in a project. These are known as the Development Switches and the Release Switches.

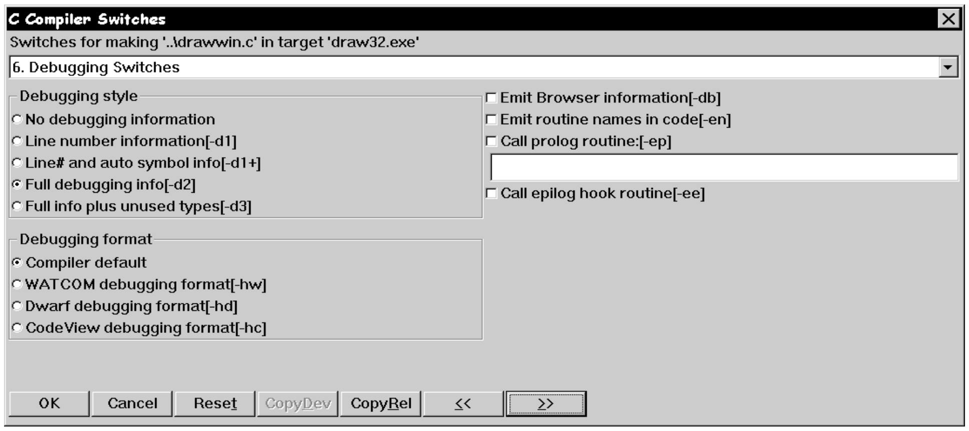
1. Right click on `DRAW???.C` and select *Source options* from the pop-up menu. Select *C Compiler Switches* from the sub-menu.

   By default, your target is placed in development mode with the debugging switches for the compiler and linker set to include debugging information. You can either set the switches in each category manually or you can copy the default Release switches using the CopyRel button. This method of setting switches is especially convenient since you can specify everything from diagnostic, optimization, and code generation switches to special linker switches all without having to memorize a cryptic switch name — you simply click next to the switches you wish to use for a particular item.

2. Scroll through the categories using the >> button until you get to:

   ```
   6. Debugging Switches
   ```

   We can see that full debugging information was used in the compile, so just click on *Cancel* to exit this screen.


C Compiler Switches

Switches for making '..\drawwin.c' in target 'draw32.exe'

6. Debugging Switches

Debugging style
- No debugging information
- Line number information[-d1]
- Line# and auto symbol info[-d1+]
- [✓ Full debugging info[-d2]]
- Full info plus unused types[-d3]

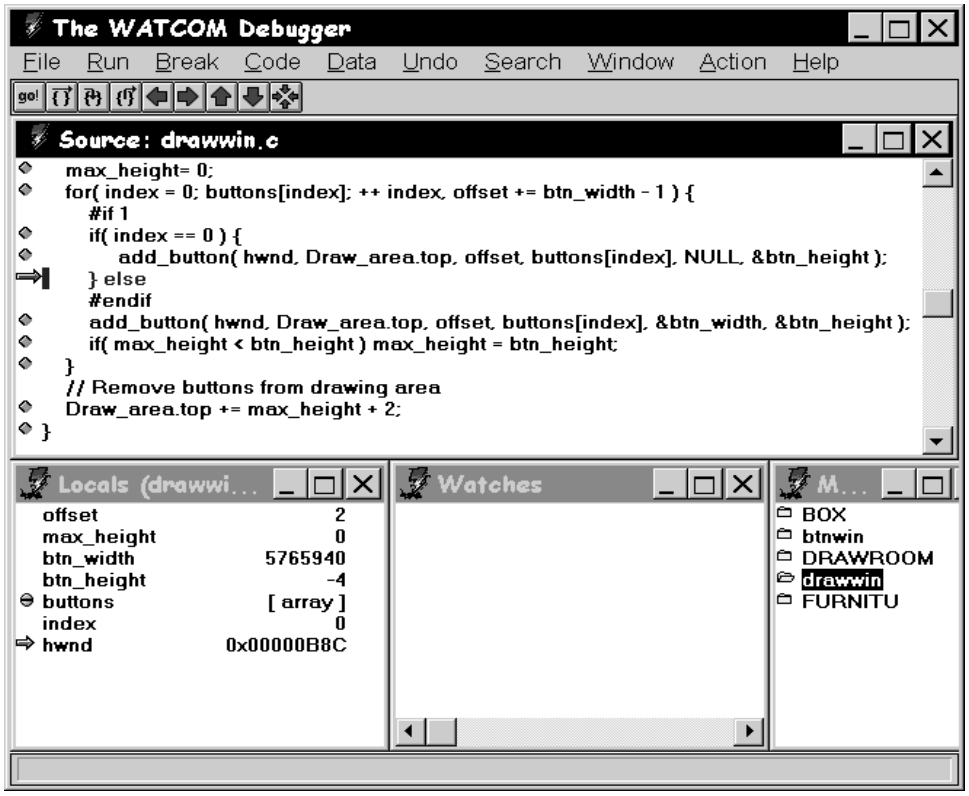
Debugging format
- [✓ Compiler default]
- WATCOM debugging format[-hw]
- Dwarf debugging format[-hd]
- CodeView debugging format[-hc]

- Emit Browser information[-db]
- Emit routine names in code[-en]
- Call prolog routine:[-ep]
- Call epilog hook routine[-ee]

Figure 8. *Setting compiler switches*

3. To invoke the debugger, pull down the *Targets* menu and choose the *Debug* item or select the "Debug the current target" icon from the toolbar.

The Watcom Debugger is designed to be as convenient and intuitive as possible, while at the same time providing a comprehensive and flexible environment for serious debugging. You can configure your environment to display exactly the information you require to be most productive. Among the windows available are source and assembly, modules, functions, calls, threads, images, watches, locals, globals, file variables, registers, 80x87 FPU, stack, I/O ports, memory display, and a log window. You can step through your source using the keys or icons on the toolbar. Execute one line at a time by stepping over calls or stepping into calls, or execute until the current function returns. Right-mouse button functionality gives context-sensitive pop-up menus.



```
    max_height= 0;
    for( index = 0; buttons[index]; ++ index, offset += btn_width - 1 ) {
        #if 1
        if( index == 0 ) {
            add_button( hwnd, Draw_area.top, offset, buttons[index], NULL, &btn_height );
        } else
        #endif
        add_button( hwnd, Draw_area.top, offset, buttons[index], &btn_width, &btn_height );
        if( max_height < btn_height ) max_height = btn_height;
    }
    // Remove buttons from drawing area
    Draw_area.top += max_height + 2;
}
```

Locals (drawwi...

```
offset                 2
max_height             0
btn_width        5765940
btn_height            -4
buttons          [ array ]
index                  0
hwnd          0x00000B8C
```

Figure 9. *The Watcom Debugger*

We know that a fault has occurred in `DRAW???.EXE,` so we will run the application and examine the state of the program when the fault occurs.

1. Click on the "go!" icon on the toolbar to begin execution of the program. The exception occurs and the source window shows the line

```
    *pwidth = bitmap.bmWidth + 5;
```

   in the function `button_size` as the last line executed before the exception. Examining the Locals window you will see that `pwidth` is a NULL pointer, hence the exception.

2. We can now move up the call stack by clicking on the "Move up the call stack" icon on the toolbar (red up arrow) to follow the program's execution. On the previous line, we see `button_size` is called from `add_button.` Moving up the call stack again, we see `add_button` is called with NULL as its fifth

parameter. An artificial error has been introduced for the purposes of this tutorial. It is located several lines back in the source file.

3. By replacing the line `#if 1` with `#if 0` we can bypass this error. Right-click on the line `#if 1` and select Show, then Line... from the pop-up menus to see the line number which must be corrected, then exit the debugger.

4. Double-click on `DRAW???.C` to load the source file into the editor.

5. Scroll down to line 217 using the keyboard or mouse, or pull down the *Edit* menu, select *Goto Line...*, and enter 217.

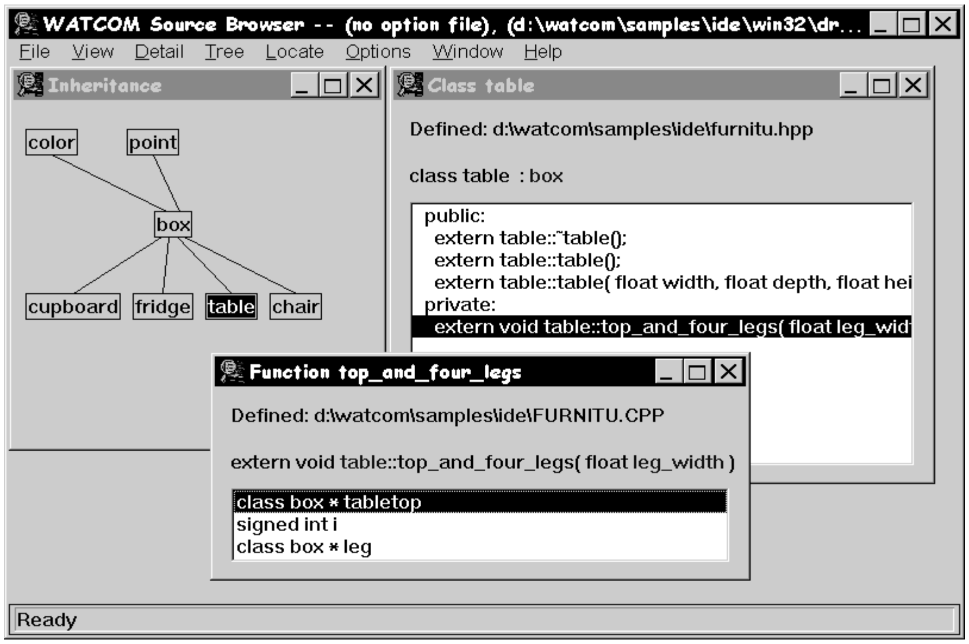
6. Replace the line `#if 1` with `#if 0` and save your changes by clicking on the "Write the current file to disk" icon or selecting *Save* from the *File* menu.

7. Return to the IDE (by clicking on it if it is visible on your screen, or by using Alt-Tab) and re-make your project.

8. Run your project to see the kitchen demo.

Using the Source Browser

Suppose you wanted to change the color of the tabletop in your application. You can use the Watcom Browser to determine the code you will need to change. The Watcom Browser provides an easy way to examine the class definitions, member functions, and inheritance trees associated with your C++ code. First, you need to instruct the compiler to emit Watcom Browser information.


WATCOM Source Browser -- (no option file), (d:\watcom\samples\ide\win32\dr...
File View Detail Tree Locate Options Window Help
Inheritance
color
point
box
cupboard
fridge
table
chair
Class table
Function top_and_four_legs
Ready

Defined: d:\watcom\samples\ide\furnitu.hpp

class table : box

```
public:
  extern table::~table();
  extern table::table();
  extern table::table( float width, float depth, float hei
private:
  extern void table::top_and_four_legs( float leg_wid
```

Defined: d:\watcom\samples\ide\FURNITU.CPP

extern void table::top_and_four_legs(float leg_width)

```
class box * tabletop
signed int i
class box * leg
```

Figure 10. *The Watcom Browser*

1. Right click on `FURNITU.CPP,` then select *Source options* from the pop-up menus. Select *C++ Compiler Switches* from the sub-menu.

2. Go to the

   `6. Debugging Switches`

   category by selecting it from the drop-down list box or by scrolling through the categories using the >> button.

3. Select `Emit Browser information [-db]` and click on OK.

4. Click the "Make all targets in the project" icon to re-make the project. The compiler will emit Browser information for `FURNITU.CPP` in a file called `DRAW???.DBR.` Now you are ready to browse the target's source.

5. Pull down the *Targets* menu and select *Browse*, or click the "Browse the current target" icon on the toolbar. The inheritance tree for the target is displayed.

6. To view details on any particular class, double-click on the item for information such as the location of the class definition, the private, public, and protected

functions of the class, and the class inheritance. Branches of the inheritance tree can be collapsed and expanded. A variety of tools are available to help you navigate your C++ source. Double-click on the `table` class.

7. Double-click on the function `top_and_four_legs()` to see the details on this function.

8. Select the variable `tabletop`, pull down the *Detail* menu, and select the *Goto Definition...* item. The Editor is invoked, loading the file `FURNITU.CPP` which contains the definition of `top_and_four_legs`.

9. Next we will make some changes to `FURNITU.CPP` in order to change the color of the tabletop. Scroll down to line 132 using the keyboard or mouse, or pull down the *Edit* menu, select *Goto Line...*, and enter 132.

10. Replace the line

    ```
    tabletop->rgb(0,255,255);
    ```

    with

    ```
    tabletop->black();
    ```

11. Save your changes by clicking on the "Write the current file to disk" icon or selecting *Save* from the *File* menu.

12. Shut down the Browser before re-making the project.

13. Return to the IDE (by clicking on it if it is visible on your screen, or by using Alt-Tab).

14. Click the "Make all targets in the project" icon to re-make the project.

Correcting an Error

An error is encountered during the make and error message(s) appear in the log window. Additional information on the error is available by selecting the error, pulling down the *Log* menu and selecting the *Help on Message* item.

1. Double-click on the error message

    ```
    furnitu.cpp (132): Error! E029: (col 15) symbol ’black’ has not
    been declared.
    ```

The offending source file (`FURNITU.CPP`) is loaded into the Editor and the cursor is positioned at the line which caused the error. Apparently, black has not been defined as a color.

2. Restart the Browser.

3. Double-click on `color` in the Inheritance window to see the member functions of the class `color.` Among our choices are `blue(),` `green(),` and `red().`

4. Press the Alt-Tab key combination to return to the Editor and replace the line

   ```
   tabletop->black();
   ```

   with

   ```
   tabletop->red();
   ```

5. Save your changes.

6. Return to the IDE and re-make the project.

7. Run the program to see the changes you have made to the tabletop.

Editing a Bitmap

You can edit bitmaps, icons, or cursors associated with your project using Watcom’s Image Editor. Double-click on a file with a `.BMP,` `.ICO,` or `.CUR` extension and the file is loaded into the Image Editor. The editor has many features to design your images, including resizing, rotation, shifting, and a utility to take a "snapshot" of another image and import it.

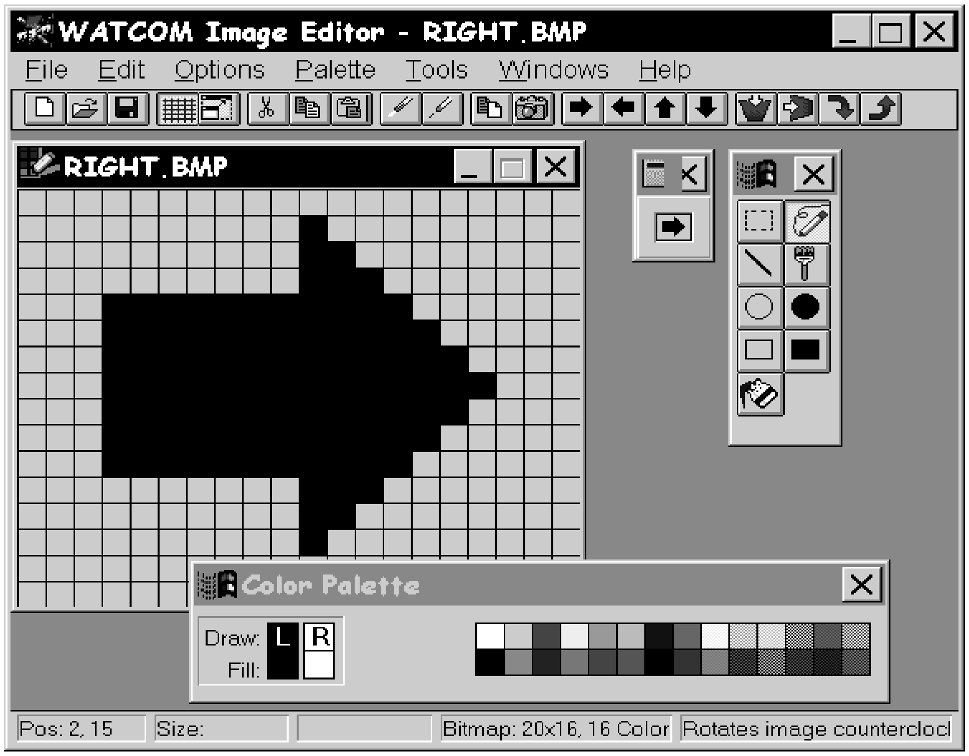

WATCOM Image Editor - RIGHT.BMP
File Edit Options Palette Tools Windows Help
RIGHT.BMP
Color Palette
Draw: L R
Fill:
Pos: 2, 15
Size:
Bitmap: 20x16, 16 Color
Rotates image countercloc

Figure 11. *The Watcom Image Editor*

Suppose you wanted to change the color of the right-arrow icon in your application.

1. If the "Folder" icon next to `.BMP` is closed, click on it to restore all the files with a `.BMP` extension to the file list.

2. Scroll the window until the file `RIGHT.BMP` is visible.

3. Double-click on `RIGHT.BMP` in the `DRAW???.EXE` target window.

4. Select the "Paint Can" icon from the Tool Palette.

5. Select a color from the Color Palette.

6. Click on the arrow.

7. Save your changes using the "Save" icon on the toolbar and exit the Image Editor.

8. Click the "Make all targets in the project" icon to rebuild the project with the change incorporated.

Editing Menus

Next, you will add source files to the list of items that make up `DRAW???.EXE.`

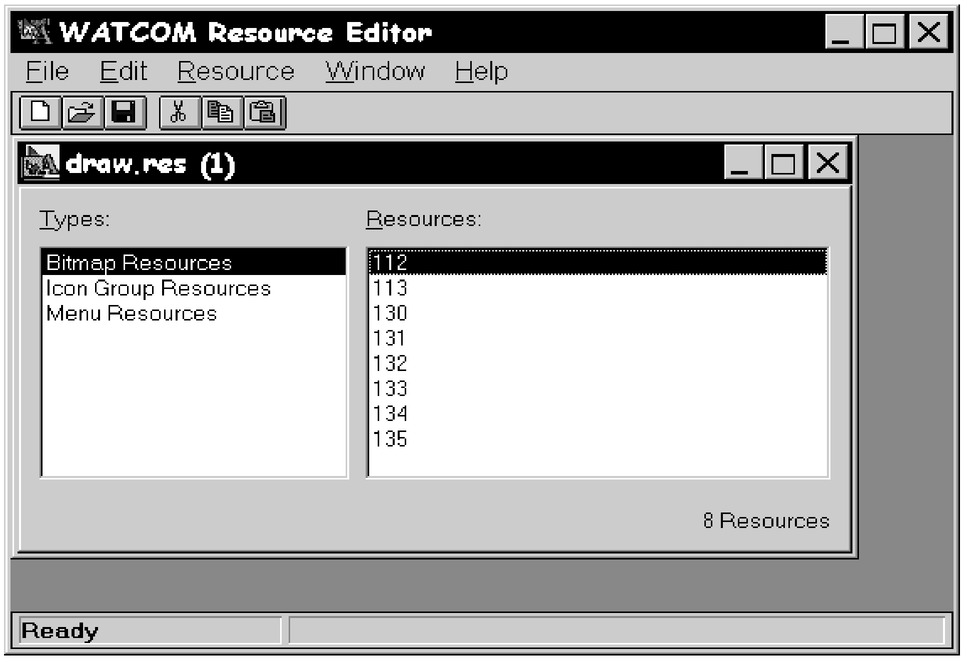
1. Pull down the *Sources* menu and select the *New Source...* item.

   Note: You can do this either by choosing from the menu bar or by positioning the mouse over the file list area and clicking the right mouse button. The IDE displays a pop-up menu from which you can choose the desired action.

2. Enter the filename `DRAW.RES` (or `DRAWOS2.RES` for OS/2). For OS/2, click OK when you have entered the source file name. For all other systems, click on Add when you have entered the source file name and then click on Close. Now we will remove the `.RC` file from the project so that our changes to the `.RES` file will not be overwritten. When an `.RC` file is present, the `.RES` file is generated from the `.RC` file.

3. Right click on `DRAW.RC` (or `DRAWOS2.RC` for OS/2), then select *Remove Source* from the pop-up menu.

4. Double-click on `DRAW.RES` (or `DRAWOS2.RES`). The Resource Editor is invoked, displaying all the available resources (in this case, icons, bitmaps, and menus).



Types:

- Bitmap Resources
- Icon Group Resources
- Menu Resources

Resources:

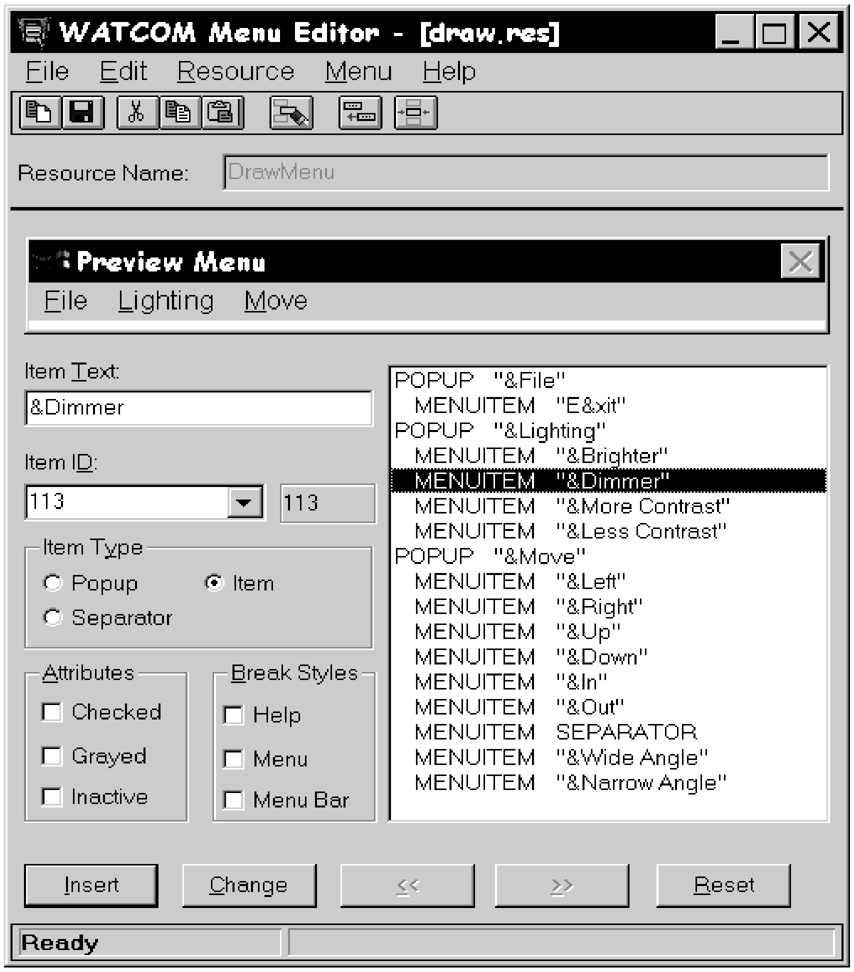
- 112
- 113
- 130
- 131
- 132
- 133
- 134
- 135

8 Resources

Figure 12. *The Watcom Resource Editor*

5. Click on "Menu Resources".

6. Double-click on "DrawMenu" in the right-hand box. This will bring up the Menu Editor. The Menu Editor displays the menus defined for the resource `DrawMenu.` You can specify pop-up menus, menu items and sub-items, text, separators, attributes, break styles, and memory flags.



Resource Name: DrawMenu

Item Text: &Dimmer

Item ID: 113 113

Item Type: ( ) Popup [✓ Item] ( ) Separator

Attributes: ☐ Checked ☐ Grayed ☐ Inactive

Break Styles: ☐ Help ☐ Menu ☐ Menu Bar

```
POPUP   "&File"
  MENUITEM   "E&xit"
POPUP   "&Lighting"
  MENUITEM   "&Brighter"
  MENUITEM   "&Dimmer"
  MENUITEM   "&More Contrast"
  MENUITEM   "&Less Contrast"
POPUP   "&Move"
  MENUITEM   "&Left"
  MENUITEM   "&Right"
  MENUITEM   "&Up"
  MENUITEM   "&Down"
  MENUITEM   "&In"
  MENUITEM   "&Out"
  MENUITEM   SEPARATOR
  MENUITEM   "&Wide Angle"
  MENUITEM   "&Narrow Angle"
```

Figure 13. *The Watcom Menu Editor*

7. Click on `MENUITEM "&Dimmer"` in the item list window.

8. In the "Item Text" window change the item to `&Darker` and then click on the "Change" button.

9. Select *Update* from the *File* menu or click on the "Update the file with this menu" icon.

10. Exit the Menu Editor.

11. Now, select *Save* from the *File* menu or click on the "Save this file" icon and exit the Resource Editor.

12. Click the "Make all targets in the project" icon to re-make the project.

Sampling and Profiling an Executable

Together, the Watcom Execution Sampler and the Watcom Execution Profiler allow you to pinpoint the areas of your code that are the most heavily used, indicating possible candidates for performance improvements.

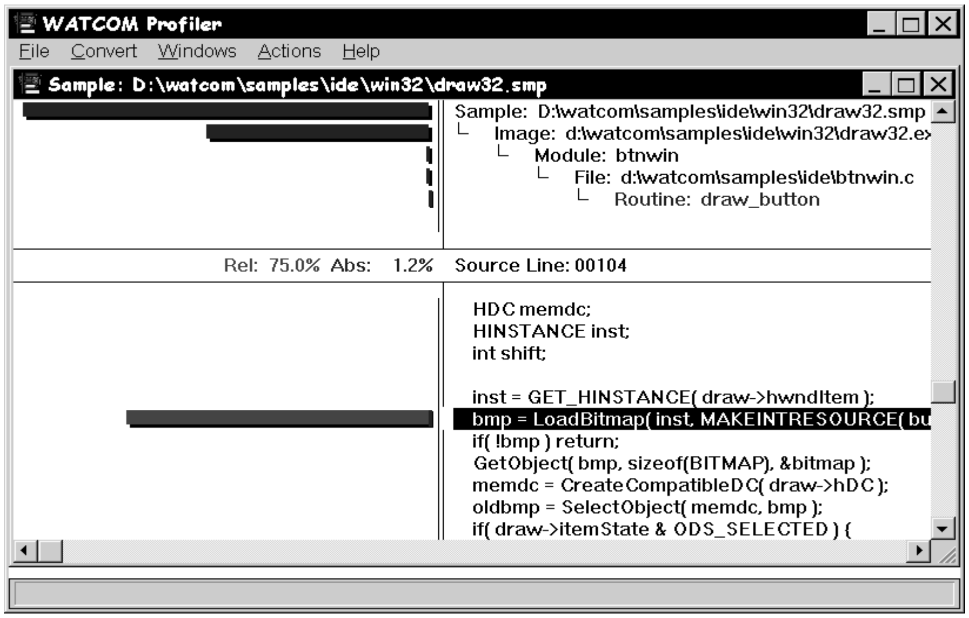
1. Click on the `DRAW???.EXE` target.

2. Select the *Sample* item from the *Targets* menu item, or click on the "Run and sample the current target" icon from the toolbar. The Watcom Execution Sampler is invoked and your application begins to execute.

3. Try rotating and resizing the image a few times. The sampler takes a "snapshot" of the code that is being executed at regular intervals. Exit the application. A samples file with extension `.SMP` is created in the current directory. This file is input for the profiler.

4. You are now ready to profile the executable. Do this by selecting *Profile* from the *Targets* menu, or by clicking on the "Profile the current target" icon on the toolbar. The profiler scans the `.SMP` file and reports the activity in the various modules of the application. The percentage of time spent in the modules is indicated as an absolute percentage (percent of total samples) and as a relative percentage (percent of samples in the `.EXE` image).

5. Double-click on the module or routine names to step down to the exact source being executed when a sample was taken. For more details, you can adjust the sampling rate of the Sampler to get a better picture of your code. To do this, exit the Profiler, pull down the *Targets* menu, and select *Target options*, then *Sample Switches...* from the pop-up menus. Specify a sampling rate such as 2 (for 2 milliseconds), click on OK, then run the Sampler and Profiler again.


WATCOM Profiler
File Convert Windows Actions Help
Sample: D:\watcom\samples\ide\win32\draw32.smp

```
Sample:  D:\watcom\samples\ide\win32\draw32.smp
└   Image:  d:\watcom\samples\ide\win32\draw32.ex
    └   Module:  btnwin
        └   File:  d:\watcom\samples\ide\btnwin.c
            └   Routine:  draw_button

Rel:  75.0%  Abs:    1.2%   Source Line: 00104

  HDC memdc;
  HINSTANCE inst;
  int shift;

  inst = GET_HINSTANCE( draw->hwndItem );
  bmp = LoadBitmap( inst, MAKEINTRESOURCE( bu
  if( !bmp ) return;
  GetObject( bmp, sizeof(BITMAP), &bitmap );
  memdc = CreateCompatibleDC( draw->hDC );
  oldbmp = SelectObject( memdc, bmp );
  if( draw->itemState & ODS_SELECTED ) {
```

Figure 14. *The Watcom Execution Profiler*

Saving the Project and Terminating the Session

You can now exit the IDE session by selecting *Exit* from the *File* menu. If you have not already saved your project, you will be prompted to do so. Choose "Yes" and the session ends.

3.3 Tutorial Review

In this tutorial, you created a project called `KITCHEN.WPJ,` which was composed of two targets: `DRAW???.EXE` and `BUTTON.LIB.` You compiled and linked it into an executable program using the WMAKE utility, the Watcom C and C++ compilers, and the Watcom Linker. You executed it both directly and under the control of the Watcom Debugger.. You browsed the source, and made changes using the text and resource editors. Finally, you sampled and profiled the application.

When you saved the project, you created the following permanent files:

- `KITCHEN.WPJ` — describes the screen layout and refers to the target files called `DRAW???.TGT` and `BUTTON.TGT.`

- `DRAW???.TGT` — describes the target executable `DRAW???.EXE` and all switches required to link it. It also describes the `.C` and `.CPP` files and switches required to compile them.

- `BUTTON.TGT` — describes the target library and all switches required to create it. It also describes the `.C` file and the switches used to build the library.

4 Documentation

The following manuals comprise the Watcom C/C++ documentation set. Printed copies of this documentation can be ordered from Watcom. When you install the software, portions of the documentation set are provided as on-line help files. Subsequent sections describe how to access this on-line help.

The following describes the titles in the Watcom C/C++ documentation set.

Watcom C/C++ User's Guide
This manual describes how to use Watcom C/C++. It contains an introduction to the compiler and a tutorial section. It also describes compiler options, precompiled header files, libraries, memory models, calling conventions, pragmas, in-line assembly, ROM based applications, and environment variables.

Watcom C/C++ Tools User's Guide
This manual describes the command line oriented tools including the compile and link utility, library manager, object file disassembler, far call optimization tool, assembler, patch utility, strip utility, make utility, and touch utility.

Watcom Graphical Tools User's Guide
This manual describes Watcom's Windows and OS/2 graphical tools including the Integrated Development Environment, Browser, Dr. Watcom, Spy, DDE Spy, Image Editor, Resource Editor, Sampler/Profiler, Resource Compiler, Heap Walker, Zoom, and Editor.

Watcom C/C++ Programmer's Guide
This manual includes 5 major sections each of which describes operating system specific development issues. The operating systems covered include extended DOS, OS/2, Windows 3.x, Windows NT/Win32s, 32-bit Windows 3.x (using Watcom's Supervisor technology), AutoCAD ADS and Novell NLMs. Topics include creating a sample program, operating system specific error messages, and debugging techniques.

Watcom C Language Reference
This manual describes the ANSI C programming language and extensions which are supported by Watcom C.

Watcom C Library Reference, Volumes 1 and 2
These manuals describe the C and graphics libraries supported by Watcom C/C++.

Watcom C++ Class Library Reference
This manual provides a comprehensive reference to the C++ class libraries provided with Watcom C/C++.

Watcom Debugger User's Guide
This manual describes the Watcom Debugger and discusses advanced debugging techniques.

Watcom Linker User's Guide
This manual describes how to use the Watcom Linker to generate executables for target systems such as extended DOS, Windows 3.x, Windows 95, Windows NT, OS/2, and Novell NLMs.

Visual Programmer User's Guide
This manual describes how to use Blue Sky's Visual Programmer, an easy to use Prototyper and C/C++ Code Generator for Windows, Win32 and Windows NT.

MFC Switch-It Module User's Guide
Blue Sky's MFC Switch-It Module (SIM) allows you to generate Microsoft Foundation Classes (MFC) C++ code for your application design(s). You prototype and test your application using Visual Programmer, then generate the code using the MFC SIM.

The following book is included in the printed documentation set. It is not available as an on-line document.

The C++ Programming Language by Bjarne Stroustrup
This book is an industry recognized authoritative standard on C++ programming. This manual begins with a tutorial to introduce the C++ concepts. The second section consists of an in-depth C++ reference guide.

4.1 Accessing On-line Documentation

The following sections describe how to access the on-line help that is available for DOS, Windows 3.x, Windows 95, Windows NT and OS/2.

On-line Documentation under DOS

The Watcom Help program, ***WHELP,*** may be used under DOS to access on-line documentation. The Watcom Help command line syntax is:

```
WHELP help_file [topic_name]
```

Notes:

1. If *help_file* is specified without an extension then ".IHP" is assumed.

2. The *topic_name* parameter is optional.

3. If *topic_name* is not specified, the default topic is "Table of Contents".

4. If *topic_name* contains spaces then it must be enclosed in quotes.

The following help files are available:

CGUIDE	***Watcom C/C++ User's Guide*** (excludes C and C++ Diagnostic Messages appendices which are available as separate help files)
CLIB	***Watcom C Library Reference***
CLR	***Watcom C Language Reference***
CMIX	***Watcom C/C++ Master Index***
CPPLIB	***Watcom C++ Class Library Reference***
ISVCPP	***Watcom C/C++ 11.0 Add-In Tools Guide***
LGUIDE	***Watcom Linker User's Guide***
PGUIDE	***Watcom C/C++ Programmer's Guide***
C_README	***Watcom C/C++ Getting Started manual***
RESCOMP	Documentation for the Watcom Resource Compiler for Windows (excerpt from the ***Watcom Graphical Tools User's Guide***)
TOOLS	***Watcom C/C++ Tools User's Guide***

WD	***Watcom Debugger User’s Guide***
WPROF	Documentation for the Watcom Execution Sampler and Watcom Execution Profiler (excerpt from the ***Watcom Graphical Tools User’s Guide***)
WCCERRS	Documentation for the Watcom C Diagnostic Messages (excerpt from the ***Watcom C/C++ User’s Guide***).
WPPERRS	Documentation for the Watcom C++ Diagnostic Messages (excerpt from the ***Watcom C/C++ User’s Guide***).

On-line Documentation under Windows 3.x, 95 and NT

On-line documentation is presented in the form of Windows Help files (".HLP" files). When the software is installed under Windows 3.x, Windows 95 or Windows NT, a number of program groups are created. You can access the on-line document by opening a program group and double-clicking on a help icon.

Watcom C/C++ Group

Getting Started	***Watcom C/C++ Getting Started***

Watcom C/C++ Tools Help Group

Accelerator Editor Help	Documentation for the Accelerator Editor (excerpt from the ***Watcom Graphical Tools User’s Guide***)
C Error Messages	Documentation for the Watcom C Diagnostic Messages (excerpt from the ***Watcom C/C++ User’s Guide***)
C++ Error Messages	Documentation for the Watcom C++ Diagnostic Messages (excerpt from the ***Watcom C/C++ User’s Guide***)
C Language Reference	***Watcom C Language Reference***
C Library Reference	***Watcom C Library Reference***
C++ Library Reference	***Watcom C++ Class Library Reference***
C/C++ Master Index	The master index for all of the Watcom C/C++ on-line help
DDE Spy Help	Documentation for the DDE Spy utility (excerpt from the ***Watcom Graphical Tools User’s Guide***)

Debugger Help	***Watcom Debugger User's Guide***
Dialog Editor Help	Documentation for the Dialogue Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Dr. Watcom Help	Documentation for Dr. Watcom (excerpt from the ***Watcom Graphical Tools User's Guide***)
Editor Help	Documentation for the Watcom Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Heap Walker Help	Documentation for the Heap Walker utility (excerpt from the ***Watcom Graphical Tools User's Guide***)
IDE Help	Documentation for the Interactive Development Environment (excerpt from the ***Watcom Graphical Tools User's Guide***)
Image Editor Help	Documentation for the Image Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Linker Guide	***Watcom Linker User's Guide***
Menu Editor Help	Documentation for the Menu Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Profiler Help	Documentation for the Watcom Execution Sampler and Watcom Execution Profiler (excerpt from the ***Watcom Graphical Tools User's Guide***)
Programmer's Guide	***Watcom C/C++ Programmer's Guide***
Resource Compiler Help	Documentation for the Resource Compiler (excerpt from the ***Watcom Graphical Tools User's Guide***)
Resource Editor Help	Documentation for the Resource Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Source Browser Help	Documentation for the Browser (excerpt from the ***Watcom Graphical Tools User's Guide***)
Spy Help	Documentation for the Spy utility (excerpt from the ***Watcom Graphical Tools User's Guide***)

String Editor Help	Documentation for the String Editor (excerpt from the ***Watcom Graphical Tools User's Guide***)
Tools Guide	***Watcom C/C++ Tools User's Guide***
User's Guide	***Watcom C/C++ User's Guide*** (excludes C and C++ Diagnostic Messages appendices which are available as separate help files)
Visual Programmer Help	***Visual Programmer User's Guide***
Visual Programmer MFC Help	***MFC Switch-it Module User's Guide***
Zoom Help	Documentation for the Zoom utility (excerpt from the ***Watcom Graphical Tools User's Guide***)

The second set of help files, listed below, forms part of the Microsoft Windows SDK.

Watcom C/C++ Additional Help Group

Add-In Tools Guide	A guide to third-party software tools/libraries
Hotspot Editor Help	
MCI Command Strings	
MFC 4.1 Library Help	
MFC 4.1 Sample Program Help	
MFC 4.1 Technical Notes	
MFC 2.52b Library Help	
MFC 2.52b Sample Program Help	
MFC 2.52b Technical Notes	
Multimedia Reference	
Pen API Reference	
Win32 API Help	

Windows API Reference

On-line Documentation under OS/2

On-line documentation is presented in the form of OS/2 Information files (".INF" files). When the software is installed under OS/2, the Watcom C/C++ folder is created. You can access the on-line document by opening the Watcom C/C++ folder and double-clicking on a help icon.

C Error Messages	Documentation for the C Diagnostic Messages (excerpt from the ***Watcom C/C++ User's Guide***)
C++ Error Messages	Documentation for the C++ Diagnostic Messages (excerpt from the ***Watcom C/C++ User's Guide***)
C Language Reference	***Watcom C Language Reference***
C Library Reference	***Watcom C Library Reference***
C++ Library Reference	***Watcom C++ Class Library Reference***
C/C++ Master Index	The master index for all of the Watcom C/C++ on-line help
CP Reference	OS/2 Control Program Reference
Debugger Help	***Watcom Debugger User's Guide***
Getting Started	***Watcom C/C++ Getting Started***
IDE Help	Documentation for the Interactive Development Environment (excerpt from the ***Watcom Graphical Tools User's Guide***)
IPFC Reference	OS/2 IPF Compiler Reference
Linker Guide	***Watcom Linker User's Guide***
PM Reference	OS/2 Presentation Manager Reference
Profiler Help	Documentation for the Watcom Execution Sampler and Watcom Execution Profiler (excerpt from the ***Watcom Graphical Tools User's Guide***)

Programmer's Guide	***Watcom C/C++ Programmer's Guide***
REXX Reference	OS/2 REXX Reference
SOM Reference	OS/2 System Object Model Reference
Source Browser Help	Documentation for the Watcom Browser (excerpt from the ***Watcom Graphical Tools User's Guide***)
Tools Guide	***Watcom C/C++ Tools User's Guide***
Tools Reference	OS/2 Toolkit Reference
User's Guide	***Watcom C/C++ User's Guide*** (excludes C and C++ Diagnostic Messages appendices which are available as separate help files)

5 Microsoft Foundation Classes

Two versions of MFC are included in this release of the compiler. Use version MFC 4.1 to create 32-bit applications. Use version MFC 2.52b to create 16-bit applications. MFC help is available by clicking on the MFC help icon in the Watcom C/C++ folder. MFC programs will not run under the Watcom 32-bit Windows Extender.

5.1 MFC 4.1

MFC 4.1 is used to create 32-bit MFC applications that run under the Win32 API (Windows NT, Windows 95) and Win32s. There is a batch file that creates the MFC 4.1 sample programs.

```
C:\WATCOM\SAMPLES\MFC\V41\MKSAMPLE.BAT
```

5.2 MFC 2.52b

MFC 2.52b is used to create 16-bit MFC applications that run under the Windows API. There is a batch file that creates the MFC 2.52b sample programs.

```
C:\WATCOM\SAMPLES\MFC\V252\MKSAMPLE.BAT
```

5.3 Debugging MFC Applications

Watcom C/C++ contains compiled libraries for MFC. For each library that is included, there is a production library and a "DEBUG" library. The debug library usually has the same name as the production library but with a "D" suffix on the file name. For example, `NRFXCC.LIB` and `NRFXCCD.LIB` are the production and debug versions of one of the libraries included in the package. The debug library is compiled with line number debugging information ("d1") and with debugging "assertions" enabled.

If you wish to have a version of the library that contains "d2" type debugging information then you must recompiled the MFC source code that is provided. Makefiles tailored for Watcom C++ are provided for building the MFC libraries.

To debug an MFC application, you must ensure the following:

1. The debug libraries must be installed.
2. If you want to see the MFC source code when debugging an MFC function, you must have installed the source code. You must also use the debugger’s "set source path" to set the source path explicitly in the debugger for the MFC source (when using libraries that were compiled by Watcom).
3. If you want to use the Watcom Browser to see MFC classes, you must select the Watcom C++ "db" option when compiling your source code.

Please note that if you wish to profile the execution of you MFC application with the Watcom Execution Profiler, you will not be able to drill down to MFC library source code unless you have recompiled the source code on your own system.

6 SOMobjects Developer Toolkit

Watcom C/C++ contains the SOMobjects Developer Toolkit for OS/2. This toolkit allows for the creation of applications that use IBM’s System Object Model (SOM). SOM is a set of libraries, utilities and conventions used to create binary class libraries.

A number of sample programs are provided with the SOMobjects Developer Toolkit for OS/2. These samples are located in the SOM\SAMPLES directory of the directory in which you installed the software. Each sample directory contains a "readme" file which describes the procedures you should follow to create and run these samples. Please read this file before attempting to create or run any of the sample programs.

The SOMobjects Developer Toolkit Publication is a complete set of manuals that describes SOM and the tools that support its use. This publication is available from IBM and includes the following:

- User’s Guide
- Programmer’s Reference
- Quick Reference
- Collection Classes Reference
- Emitter Guide and Reference
- Installation/Configuration Instructions

The SOMobjects Developer Toolkit for Windows is not included in Watcom C/C++ version 11.0.

7 Benchmarking Hints

The Watcom C/C++ compiler contains many options for controlling the code to be produced. It is impossible to have a certain set of compiler options that will produce the absolute fastest execution times for all possible applications. With that said, we will list the compiler options that we think will give the best execution times for most applications. You may have to experiment with different options to see which combination of options generates the fastest code for your particular application.

The recommended options for generating the fastest 16-bit Intel code are:

Pentium Pro	/oneatx /oh /oi+ /ei /zp8 /6 /fpi87 /fp6
Pentium	/oneatx /oh /oi+ /ei /zp8 /5 /fpi87 /fp5
486	/oneatx /oh /oi+ /ei /zp8 /4 /fpi87 /fp3
386	/oneatx /oh /oi+ /ei /zp8 /3 /fpi87 /fp3
286	/oneatx /oh /oi+ /ei /zp8 /2 /fpi87 /fp2
186	/oneatx /oh /oi+ /ei /zp8 /1 /fpi87
8086	/oneatx /oh /oi+ /ei /zp8 /0 /fpi87

The recommended options for generating the fastest 32-bit Intel code are:

Pentium Pro	/oneatx /oh /oi+ /ei /zp8 /6 /fp6
Pentium	/oneatx /oh /oi+ /ei /zp8 /5 /fp5
486	/oneatx /oh /oi+ /ei /zp8 /4 /fp3
386	/oneatx /oh /oi+ /ei /zp8 /3 /fp3

The "oi+" option is for C++ only. Under some circumstances, the "ob" and "ol+" optimizations may also give better performance with 32-bit Intel code.

Option "on" causes the compiler to replace floating-point divisions with multiplications by the reciprocal. This generates faster code (multiplication is faster than division), but the result may not be the same because the reciprocal may not be exactly representable.

Option "oe" causes small user written functions to be expanded in-line rather than generating a call to the function. Expanding functions in-line can further expose other optimizations that couldn't otherwise be detected if a call was generated to the function.

Option "oa" causes the compiler to relax alias checking.

Option "ot" must be specified to cause the code generator to select code sequences which are faster without any regard to the size of the code. The default is to select code sequences which strike a balance between size and speed.

Option "ox" is equivalent to "obiklmr" and "s" which causes the compiler/code generator to do branch prediction ("ob"), expand intrinsic functions in-line ("oi"), enable control flow prologues and epilogues ("ok"), perform loop optimizations ("ol"), generate 387 instructions in-line for math functions such as sin, cos, sqrt ("om"), reorder instructions to avoid pipeline stalls ("or"), and to not generate any stack overflow checking ("s"). Option "or" is very important for generating fast code for the Pentium and Pentium Pro processors.

Option "oh" causes the compiler to attempt repeated optimizations (which can result in longer compiles but more optimal code).

Option "oi+" causes the C++ compiler to expand intrinsic functions in-line (just like "oi") but also sets the *inline_depth* to its maximum (255). By default, *inline_depth* is 3. The *inline_depth* can also be changed by using the C++ `inline_depth` pragma.

Option "ei" causes the compiler to allocate at least an "int" for all enumerated types.

Option "zp8" causes all data to be aligned on 8 byte boundaries. The default is "zp2" for the 16-bit compiler and "zp8" for 32-bit compiler. If, for example, "zp1" packing was specified then this would pack all data which would reduce the amount of data memory required but would require extra clock cycles to access data that is not on an appropriate boundary.

Options "0", "1", "2", "3", "4", "5" and "6" emit Intel code sequences optimized for processor-specific instruction set features and timings. For 16-bit Intel applications, the use of these options may limit the range of systems on which the application will run but there are execution performance improvements.

Options "fp2", "fp3", "fp5" and "fp6" emit Intel floating-point operations targetted at specific features of the math coprocessor in the Intel series. For 16-bit Intel applications, the use of these options may limit the range of systems on which the application will run but there are execution performance improvements.

Option "fpi87" causes in-line Intel 80x87 numeric data processor instructions to be generated into the object code for floating-point operations. Floating-point instruction emulation is not included so as to obtain the best floating-point performance in 16-bit Intel applications.

For 32-bit Intel applications, the use of the "fp5" option will give good performance on the Intel Pentium but less than optimal performance on the 386 and 486. The use of the "5" option will give good performance on the Pentium and minimal, if any, impact on the 386 and 486. Thus, the following set of options gives good overall performance for the 386, 486 and Pentium processors.

```
/oneatx /oh /oi+ /ei /zp8 /5 /fp3
```

8 Additional Redistribution Rights

Please read carefully the information in the following sections if you plan to distribute your application to others.

8.1 Redistributable Components

Subject to the terms and conditions of the Powersoft Language Products Software License Agreement, in addition to any Redistribution Rights granted therein, you are hereby granted a non-exclusive, royalty-free right to reproduce and distribute the Components specified below provided that (a) it is distributed as part of and only with your software product; (b) you not suppress, alter or remove proprietary rights notices contained therein; and (c) you indemnify, hold harmless and defend Powersoft and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product.

The Tenberry Software (formerly Rational Systems, Inc.) and Watcom Components are:

- DOS4GW.EXE Copyright © 1990-1994 Tenberry Software, Inc.
- RMINFO.EXE Copyright © 1987-1993 Tenberry Software, Inc.
- PMINFO.EXE Copyright © 1987-1993 Tenberry Software, Inc.
- PRIVATXM.EXE Copyright © 1991 Tenberry Software, Inc.
- DOS4GW.DOC Copyright © 1991-1995 Tenberry Software, Inc.
- WEMU387.386 Copyright © 1991-2000 Sybase, Inc.

The Microsoft Components from the Microsoft Windows SDK version 3.1, all Copyright © 1992 Microsoft Corporation, are:

- COMMDLG.DLL
- COMMDLG.DAN
- COMMDLG.DUT
- COMMDLG.FIN

- COMMDLG.FRN
- COMMDLG.GER
- COMMDLG.ITN
- COMMDLG.NOR
- COMMDLG.POR
- COMMDLG.SPA
- COMMDLG.SWE
- DDEML.DLL
- LZEXPAND.DLL
- OLECLI.DLL
- OLESVR.DLL
- PENWIN.DLL
- SHELL.DLL
- STRESS.DLL
- TOOLHELP.DLL
- VTD.386
- VER.DLL
- DIB.DRV
- EXPAND.EXE
- MARKMIDI.EXE
- REGLOAD.EXE
- WINHELP.EXE
- WINHELP.HLP

8.2 OS/2 Toolkit

You may distribute components from the OS/2 Toolkit included in the package which are header files and include files and you may modify and distribute components from the OS/2 Toolkit included in the package which are sample programs provided that such header files, include files and sample programs are distributed only for the purposes of developing, using, marketing and distributing application programs written to the OS/2 application programming interface. Further, with respect to sample programs, each copy of any portion thereof or any derivative work which is so distributed must include a copyright notice as follows:

```
(c) Copyright (your company name) (year). All Rights Reserved.
```

8.3 IBM SOMobjects

You may sublicense the IBM SOMobjects Kernel and Workstation DSOM, Version 2.0 for OS/2 and for Windows but only if incorporated as part of your application and only if: your application adds significant function to the function of the IBM SOM and DSOM components; you retain in any copies made of the IBM SOM and DSOM components all IBM copyright and proprietary rights notices contained therein. Further, you include a copyright notice in the following format on the media label attached to any copies of your application which includes the IBM SOM and DSOM components:

```
(c) Copyright (your company name) and others (date).  All Rights Reserved.
```

you indemnify Powersoft and International Business Machines Corporation, including their subsidiaries, from and against any claim arising from the distribution of your application or otherwise arising hereunder excluding any claims that the IBM SOM and DSOM components infringe a U.S. copyright held by a third party; your application and the IBM SOM and DSOM components are sublicensed pursuant to a written license agreement between you and your customer which conforms substantially to the terms and conditions of the Powersoft Language Products Software License Agreement (printed elsewhere in this document) together with the additional terms and conditions set forth herein. Without limiting the generality of the foregoing, said license agreement shall indicate that you are the licensor and shall contain provisions limiting Powersoft's and IBM liability to the same extent as they are limited in the Powersoft Language Products Software License Agreement and these additional terms and conditions.

You may sublicense to your customers the rights to reproduce and distribute the IBM SOM and DSOM components provided such sublicense is in writing and imposes terms and conditions upon your customers which are no less restrictive as are imposed upon you as set forth in the Powersoft Language Products Software License Agreement (printed elsewhere in this document) together with the additional terms and conditions set forth herein.

8.4 Microsoft MFC, Win32s

You may sublicense the Microsoft Foundation Class Libraries and Win32s (collectively called the "MS CODE") but only in object-code form and solely in conjunction with your application developed using Watcom C/C++ and only if: your application and/or any documentation bears copyright notices sufficient to protect Microsoft's copyright in the MS CODE; the MS CODE is not incorporated into any product which would enable such MS CODE to be used in conjunction with, or to develop application programs for non-Microsoft operating systems (This restriction does not apply to MS CODE which runs or can be run on non-Microsoft operating systems without modification), you indemnify Powersoft and Microsoft from and against any claim arising from the distribution of your application or otherwise arising hereunder excluding any claims that the MS CODE infringes a U.S. copyright held by a third party; your application and the MS CODE are sublicensed pursuant to a written license agreement between you and your customer which conforms substantially to the terms and conditions of the Powersoft Language Products Software License Agreement (printed elsewhere in this document) together with the additional terms and conditions set forth herein. Without limiting the generality of the foregoing, said license agreement shall indicate that you are the licensor and shall contain provisions limiting Powersoft’s and Microsoft’s liability to the same extent as they are limited in the Powersoft Language Products Software License Agreement and these additional terms and conditions. You may also modify the MFC components of the MS CODE and reproduce and distribute such modifications in object-code form as part of your application.

9 Release Notes for Watcom C/C++ 11.0

There are a number of enhancements and changes in this new version of Watcom C/C++. The following sections outline most of the major changes. You should consult the User's Guide for full details on these and other changes made to the compiler and related tools. You should check the next section to determine if you need to recompile your application.

9.1 Changes in 11.0 that may Require Recompilation

Do not attempt to mix object code generated by earlier versions of the compilers with object code generated by this release or with the libraries provided in this release.

A new C++ object model has been implemented. If you have undefined references to `__wcpp_3_*` names, you have old object code. If you have undefined references to `__wcpp_4_*,` you have old libraries and new object code.

clock() The ***clock*** function accuracy has changed from 100 ticks per second to 1000 ticks per second (i.e., ***CLOCKS_PER_SEC*** has changed). Source code that uses the ***clock*** function and ***CLOCKS_PER_SEC*** in its calculations must be recompiled before linking the application with new libraries.

9.2 Major Differences from Version 10.6

The following sections summarize the major differences from the previous release of Watcom C/C++.

- In general, we have improved Microsoft compatibility in our compilers (more warnings instead of errors, support for MS extensions, etc.) and tools.

- Some of the Win32 and 32-bit OS/2 versions of our tools are now available in DLL form.

```
EXE      DLL        Description
------   -------    -----------------------
wcc      wccd       16-bit x86 C compiler
wcc386   wccd386    32-bit x86 C compiler
wpp      wppdi86    16-bit x86 C++ compiler
wpp386   wppd386    32-bit x86 C++ compiler
wlink    wlink      Watcom Linker
wlib     wlibd      Watcom Library Manager
```

This provides better performance when using the Integrated Development Environment or Watcom Make. See the description of the `!loaddll` preprocessing directive in Watcom Make for more information.

Changes to the C++ Compiler for 11.0

- The C++ compiler now optimizes empty base-classes to occupy zero storage in the derived class memory layout. The C++ Working Paper recently allowed this optimization to be performed by conforming implementations. Furthermore, the optimization has speed and size benefits. There are certain classes of (broken) C++ programs that may not function properly with the new optimization. If you explicitly memset() an empty base class, you will be clearing memory that you may not expect to be cleared since the "zero sized" base class in actual fact shares storage with the first member of the derived class. A memset() of the entire derived class is fine though.

- We have added support for the `mutable` keyword which is used to indicate data members that can be modified even if you have a *const* pointer to the class.

  Example:
```
class S {
    mutable int x;
    void foo() const;
};

void S::foo() const {
    x = 1;  // OK since it is mutable
}
```

- We have added support for the `bool` type along with `true` and `false.`

- We have added support for the `explicit` attribute. It marks a constructor so that it will not be considered for overloading during implicit conversions.

Example:
```
struct S {
    explicit S( int );
};

S v = 1;    // error; cannot convert ’int’ to ’S’
```

Suppose the class was changed as follows:

Example:
```
struct S {
  explicit S(int );
  S( char );
};

S v = 1;  // OK; S( char ) is called
```

The fact that `S(int)` is not considered leaves `S(char)` as the only way to satisfy the implicit conversion.

• We have added support for namespaces.

```
namespace x {
    // anything that can go in file-scope
}
namespace {
    // anything in here is local to your module!
}
```

In the above example, you can access names in the namespace "x" by "x::" scoping. Alternatively, you can use the "using namespace x" statement (thereby eliminating the need for "x::" scoping). You can include a part of the namespace into the current scope with the "using x::member" statement. (also eliminating the need for "x::" scoping).

1. Namespaces eliminate the hand mangling of names. For example, instead of prefixing names with a distinguishing string like "XPQ_" (e.g., XPQ_Lookup), you can put the names in a namespace called "XPQ".
2. Namespaces allow for private names in a module. This is most useful for types which are used in a single module.
3. Namespaces encourage the meaningful classification of implementation components. For example, code-generation components might reside in a namespace called "CodeGen".

• We have added support for RTTI (Run-Time Type Information).

• We have added support for the new C++ cast notation. It allows you to use less powerful casts that the all powerful C-style cast and to write more meaningful code. The idea is to eliminate explicit casts by using a more meaningful new-style cast. The new C++ casts are:

reinterpret_cast *< type-id >(expr)*
const_cast *< type-id >(expr)*
static_cast *< type-id >(expr)*
dynamic_cast *< type-id >(expr)* (part of RTTI)

• We have improved (faster) pre-compiled header support.

• We have added "long long" (64-bit floating-point) support in the form of a new *__int64* type.

• The default structure packing was changed from "zp1" to "zp2" in the 16-bit compiler and from "zp1" to "zp8" in the 32-bit compiler.

• The default type of debugging information that is included in object files is "Dwarf". It used to be "Watcom".

• A new double-byte string processing option has been added (zkl). When this option is specified, the local or current code page character set is used to decide if the compiler should process strings as if they might contain double-byte characters.

Changes to the C Compiler for 11.0

• We have improved (faster) pre-compiled header support.

• We have added "long long" (64-bit floating-point) support in the form of a new *__int64* type.

• The default structure packing was changed from "zp1" to "zp2" in the 16-bit compiler and from "zp1" to "zp8" in the 32-bit compiler.

• The default type of debugging information that is included in object files is "Dwarf". It used to be "Watcom".

• A new double-byte string processing option has been added (zkl). When this option is specified, the local or current code page character set is used to decide if the compiler should process strings as if they might contain double-byte characters.

Changes to the Code Generator for 11.0

• We support Microsoft-compatible in-line assembly formats using the "_asm" keyword.

• A new optimization, "branch prediction", has been added. This optimization is enabled by the "ob" or "ox" compiler options. The code generator tries to increase the density of cache use by predicting branches based upon heuristics (this optimization is especially important for Intel’s Pentium Pro).

• We have added Multi-media Extensions (MMX) support to the in-line assemblers.

• We have added "long long" (64-bit floating-point) support in the form of a new *__int64* type.

Changes to the Compiler Tools for 11.0

• The Watcom Linker supports incremental linking.

• The Watcom Linker can now process COFF and ELF format object files, as well as OMF et al. The Watcom Linker can now read both AR-format (Microsoft compatible) libraries and old-style OMF libraries.

• Support for creating 16-bit DOS overlaid executables has been removed from the linker.

• The Watcom Library Manager (WLIB) can now process COFF and ELF format object files, as well as OMF et al. The Watcom Library Manager can now read/write both AR-format (Microsoft compatible) libraries and old-style OMF libraries. The default output format is AR-format and this can be changed by switches. The Watcom Library Manager can output various format import libraries.

• We have added Multi-media Extensions (MMX) support to the Watcom Assembler (WASM).

• A new version of the Watcom Disassembler (WDIS) is included. It can process ELF, COFF or OMF object files and ELF, COFF or PE format (Win32) executables.

The old disassembler (WDISASM) has been retired and is not included in the package.

• We have added new tool front-ends that emulate Microsoft tools. These are:

- nmake
- cl
- link
- lib
- rc
- cvtres

These programs take the usual Microsoft arguments and translate them, where possible, into equivalent Watcom arguments and spawn the equivalent Watcom tools.

- Watcom Make now processes Microsoft format makefiles when the "ms" option is used.

Changes to the C/C++ Libraries for 11.0

- We have added multi-byte and wide character (including UNICODE) support to the libraries.

- We include run-time DLLs for the C, Math and C++ Libraries.

- We have added Multi-media Extensions (MMX) support to the libraries.

- The following new functions were added to the library...

  multi-byte functions

- The ***clock*** function accuracy has changed from 100 ticks per second to 1000 ticks per second (i.e., ***CLOCKS_PER_SEC*** has changed).

- A "commit" flag ("c") was added to the fopen() *mode* argument.

- The global translation mode flag default is "text" unless you explicitly link your program with `BINMODE.OBJ.`

- Processing of the "0" flag in the format string for the printf() family of functions has been corrected such that when a precision is specified, the "0" flag is ignored.

Example:
```
printf( "%09.3lf\n", 1.34242 ); // "0" flag is ignored
printf( "%09lf\n", 1.34242 ); // "0" flag is not
ignored
```

• Support for printing *__int64* values was added to ***printf*** and related functions.

• Support for scanning *__int64* values was added to ***scanf*** and related functions.

• The Win32 ***_osver*** variable was added to the library.

• The Win32 ***_winmajor, _winminor*** and ***_winver*** variables were added to the library.

Changes to the DOS Graphics Library for 11.0

• The graphics library now performs the VESA test *before* testing for vendor specific graphics cards. This fix is intended to broaden the number of graphics cards that are supported.

Changes in Microsoft Foundation Classes Support for 11.0

• Version 4.1 of the 32-bit MFC is included in the package.

• Version 2.52b of the 16-bit MFC is included in the package.

Changes in Microsoft Win32 SDK Support for 11.0

• The Win32 SDK is supported for Windows 95 and Windows NT platforms.

Changes in Blue Sky's Visual Programmer for 11.0

• A new 32-bit version of Visual Programmer is included in the package. This version runs on 32-bit Windows 95 and NT. The 16-bit version of Visual Programmer is no longer included in the package.

• You can generate 16-bit applications with it, but you must be careful to avoid using Win95 controls.

• This new version fixes all known bugs in the previous version.

9.3 Changes in 10.6 that may Require Recompilation

_diskfree_t The struct members of the _diskfree_t structure has been changed from UNSIGNED SHORTs to UNSIGNED INTs. This is to deal with possible HPFS partitions whose size will overflow a short, as well as Microsoft compatibility.

clock() The ***clock*** function accuracy has changed from 100 ticks per second to 1000 ticks per second (i.e., ***CLOCKS_PER_SEC*** has changed). Source code that uses the ***clock*** function and ***CLOCKS_PER_SEC*** in its calculations must be recompiled before linking the application with new libraries.

9.4 Major Differences from Version 10.5

The following sections summarize the major differences from the previous release of Watcom C/C++.

Windows 95 Help File Format

We have included Windows 95 format help files.

Changes to the C++ Compiler in 10.6

We have improved Microsoft compatibility so that Win32 SDK and MFC header files can be compiled without change. The following changes were required to support Win32 SDK header files.

- We recognize the single underscore versions of `__stdcall`, `__inline`, and `__fastcall` keywords.

- The `_fastcall` and `__fastcall` keywords are scanned but ignored since they refer to a particular Microsoft code generation technique. Watcom’s generated code is always "fast".

The following changes were required to support MFC source code.

- When /bt=DOS is specified, define `_DOS`.

- When /bt=WINDOWS is specified, define `_WINDOWS`.

- When /m[s|m|c|l|h] is specified, define `__SW_M[S|M|C|L|H]` and `_M_I86[S|M|C|L|H]M.`

The compiler now supports the C++ Standard Template Library (STL). This library is available at the ftp site "butler.hpl.hp.com". When compiling applications that use the STL, you must use the "hd" compiler option for debugging info (the "hw" option causes too much debug information to be generated).

Changes to the C Compiler in 10.6

We have improved Microsoft compatibility so that Win32 SDK and MFC header files can be compiled without change. The following changes were required to support Win32 SDK header files.

- Support for the single underscore version of the `__stdcall` keyword.

- When /bt=DOS is specified, define `_DOS.`

- When /bt=WINDOWS is specified, define `_WINDOWS.`

The following changes were required to support SDK sample code.

- You can specify calling convention information in a function prototype and you do not have to specify the same information in the definition. (Note: This is required by the OS/2 Warp SDK samples.)

- Structured exception handling is supported (`__try,` `__except` and `__finally` keywords).

- Allow initialization of automatic array/struct data using variables and function calls.

Changes to the C Library in 10.6

The following new functions were added to the library.

_getw	read int from stream file
_putw	write int to stream file

The ***clock*** function accuracy has changed from 100 ticks per second to 1000 ticks per second (i.e., ***CLOCKS_PER_SEC*** has changed).

Changes in Microsoft Foundation Classes Support for 10.6

- Version 3.2 of the 32-bit MFC is included in the package.
- Version 2.52b of the 16-bit MFC is included in the package.

Changes to the Image Editor in 10.6

- Support has been added for 256 colour bitmaps.
- Support has been added for 16 X 16 icons.
- Support has been added for 48 X 48 icons.

Changes to the Dialog Editor in 10.6

- Support has been added for Windows 95 controls.
- Support has been added for adding new control styles to existing controls.
- Support has been added for new dialog styles.
- Support has been added for allowing help IDs to be specified in dialog and control statements.
- Support has been added for generating new resource statements in .RC files.

Changes to the Resource Editor in 10.6

- Support has been added for new Windows 95 DIALOGEX resource type.
- Support has been added for generating new DIALOGEX resource statements in .RC files.

Changes to the Resource Compiler in 10.6

- Support has been added for extended styles for dialogs.
- Support has been added for the RCINCLUDE keyword.

9.5 Major Differences from Version 10.0

- New installation program
- Visual Programmer for Windows (MFC) applications
- MFC 3.0 support
- Native C++ exception handling support
- Improved language compatibility with Microsoft
- Browser can now be used to browse C code
- OS/2 3.0 Warp support
- Toolkit for OS/2 1.3
- Windows NT 3.5 support
- Toolkit for Windows NT 3.5
- Windows 95 (Chicago) support
- Source Revision Control System hooks in editor
- TCP/IP remote debug servers for OS/2 and Windows NT/95

In addition to these new features, we have also made a number of improvements to the software.

1. The editor is more tightly integrated with the IDE.
2. It is now easier to select your own favourite editor from the IDE.

3. The keyboard interface in the Integrated Development Environment (IDE) has been improved.

4. The "fr" option, which is supported by the compilers & assembler, can be used to name the error file drive, path, file name and/or extension.

5. We have added the "t<number>" option to the C++ compiler to set the number of spaces in a tab stop (for column numbers in error messages).

6. The C compiler now supports @filename on the command line like the C++ compiler currently does.

7. The "__stdcall" linkage convention has changed. All C symbols (extern "C" symbols in C++) are now suffixed by "@nnn" where "nnn" is the sum of the argument sizes (each size is rounded up to a multiple of 4 bytes so that char and short are size 4). When the argument list contains "...", the "@nnn" suffix is omitted. This was done for compatibility with Microsoft. Use the "zz" option for backwards compatibility.

8. The 32-bit "__cdecl" linkage convention has changed. Watcom C/C++ 10.0 __cdecl did not match the Microsoft Visual C++ __cdecl in terms of the binary calling convention; Visual C++ saves EBX in a __cdecl function but Watcom C/C++ 10.0 modified EBX. Watcom C/C++ has been changed to match Visual C/C++.

   If you wrote a "__cdecl" function in an earlier version of Watcom C/C++, the EBX register was not saved/restored. Starting with release 10.5, the EBX register will be saved/restored in the prologue/epilogue of a "__cdecl" function.

   Another implication of this change is that "__cdecl" functions compiled with an earlier version of Watcom C/C++ don’t match the calling conventions of the current version. The solution is either to recompile the functions or to define a "__cdecl_old" pragma that matches the old calling conventions.

```
#pragma aux __cdecl_old "_*" \
            parm caller [] \
            value struct float struct routine [eax]
\
            modify [eax ebx ecx edx];

#pragma aux (__cdecl_old) foo;

extern int foo( int a, int b );

void main()
{
    printf( "%d\n", foo( 1, 2 ) );
}
```

9. We now allow:

   ```
   extern "C" int __cdecl x;
   ```

   It must be `extern "C"` for `__cdecl` to take effect since variables have their type mangled into the name for "C++" linkage.

10. In C++, we have removed the warning for "always true/false" expressions if the sub-expressions are constant values.

11. We have added support for:

    ```
    #pragma pack(push,4);
    #pragma pack(push);
    #pragma pack(pop)
    ```

12. We have added support for:

    ```
    #pragma comment(lib,"mylib.lib")
    ```

    which has the same semantics as:

    ```
    #pragma library( "mylib.lib" )
    ```

13. We have added support for expanding macros in the code_seg/data_seg pragmas:

```
#define DATA_SEG_NAME "MYDATA"
#define CODE_SEG_NAME "MYCODE"

#pragma data_seg( DATA_SEG_NAME )
int x = 3;

#pragma code_seg( CODE_SEG_NAME )
int fn() {
    return x;
}
```

14. We have fixed the 16-bit compiler so that it matches the Microsoft 16-bit C compiler for the following cases:

    • If a pascal function is defined when compiling for Windows 3.x, use the fat Windows 3.x prologue in the function.

    • If a cdecl function is defined when compiling for Windows 3.x, use the fat Windows 3.x prologue in the function.

15. We have fixed the compiler so that

    ```
    #include </dir/file.h>
    ```

    works as expected (it was searching along the INCLUDE path only).

16. In C++, we have fixed a problem where an import was generated in the object file for a virtual function call. This will reduce the size of executables under certain circumstances.

17. In C++, we have removed the prohibition of pointer to array of unknown size declarations.

    Example:
    ```
    int (*p)[];
    ```

18. In C++, we have fixed the diagnosis of lexical problems during macro expansion to remove spurious warnings.

Example:
```
#define stringize( x )  #x

stringize( 2131231236172637126371273612763612731 )
```

19. We have corrected the check for too many bytes in #pragma for assembler style aux #pragmas.

20. Undeclared class names in elaborated class specifiers are now declared in the nearest enclosing non-class scope. Undeclared classes are also allowed in arguments now.

    Example:
    ```
    struct S {
        // used to declared ::S::N but now declares ::N
        struct N *p;
    };

    void foo( struct Z *p );    // declares ::Z
    ```

21. We have fixed unduly harsh restriction on virtual ...-style functions. They are now allowed in single inheritance hierarchies as long as the return type is not changed when the virtual function is overridden. In multiple inheritance hierarchies, an implementation restriction is still present for generating a 'this' adjustment thunk for virtual functions.

22. We have fixed line number information for multi-line statement expressions in some weird cases.

23. We have fixed function template parsing of user-defined conversions that use an uninstantiated class in their operator name.

    Example:
    ```
    void ack( int );

    template <class T>
        struct S {
            S( T x )
            {
                ack( x );
            }
        };
    ```

```
    template <class T>
        struct W {
            operator S<T>();
        };

    template <class T>
        W<T>::operator S<T>() {
            return 0;
        }
```

24. We have fixed a compiler problem that caused a linker warning "lazy reference for <virtual-fn> has different default resolutions" in cases where the compiler or programmer optimized virtual function calls to direct calls in modules that also contained virtual calls.

    Example:

```
    T.H
        struct S {
            virtual int foo() { return __LINE__; }
        };
        struct T : S {
            virtual int foo() { return __LINE__; }
        };

    T1.CPP
        #include "t.h"
        struct Q : T {
            virtual int foo() { return S::foo() +
    __LINE__; }
        };

        void foo( T *p )
        {
            Q y;
            y.foo();
            p->foo();
        }
```

```
T2.CPP
    #include "t.h"

    void foo( T *p );

    void ack( T *p ) {
        p->foo();
        foo(p);
    }

    main() {
        T q;
        ack( &q );
    }
```

25. When a class value is returned and is immediately (in the same expression) used to call a member function, the value may not be stored in memory.

    Work around: introduce a temporary

    Example:

```
struct S {
    int v;
    int member();
};

S foo();

void example( void )
{
    // foo().member();  // replace this line with:
    S temp = foo();
    temp.member();
}
```

26. Throwing pointers to functions did not work when the size of a function pointer is greater than the size of a data pointer.

    Work around: place the function pointer in a class and throw the class object.

27. We have fixed default argument processing for const references to an abstract class. The following example would not compile properly:

Example:
```
struct A {
    virtual int foo() = 0;
};

A &foo();

void ack( A const &r = foo() );

void bar() {
    ack();
}
```

28. We have made "DllMain" default to extern "C" linkage for Microsoft Visual C++ compatibility.

29. We have duplicated a Microsoft Visual C++ extension that was required to parse the Windows 95 SDK header files.

    Example:
    ```
    typedef struct S {
    } S, const *CSP;
         ^^^^^- not allowed in ANSI C or current WP for
    C++
    ```

30. We now do not warn about starting a nested comment if the comment is just about to end.

    We also fixed the code that figures out where a comment was started so that a nested comment warning is more helpful.

    Example:
    ```
            /*//////////*/
                       ^-
    ```

31. We have fixed a problem where extra informational notes were not being printed for the error message that exceeded the error message limit.

Example:
```
// compile -e2
struct S {
    void foo();
};

void foo( S const *p )
{
    p->foo();
    p->foo();
    p->foo();
    p->foo();
}
```

32. We have fixed a problem where the line number for an error message was incorrect.

    Example:
    ```
    struct S {
        void foo() const;
        void bar();
    };

    void S::foo() const
    {
        bar();

        this->bar();

    }
    ```

33. We have fixed output of browser information for instantiated function template typedefs.

34. We have upgraded the C++ parser so that casts and member pointer dereferences can appear on the left hand side of the assignment expression without parentheses.

    Example:
    ```
    p->*mp = 1;
    (int&)x = 1;
    ```

35. In several cases, when a function return or a construction was immediately dotted in an expression, the generated code was incorrect:

Example:

```
struct S {
  int x;
  int foo();
};

extern S gorf();

void bar()
{
    gorf().foo();
}
```

The work around was to break the statement in two:

Example:

```
S temp = gorf();
temp.foo();
```

36. In several cases, when a function return or a construction was immediately addressed in an expression, the generated code was incorrect:

Example:

```
struct S {
  int x;
};

extern void fun( S* );

extern S gorf();

void bar()
{
    fun( &gorf() );
}
```

The work around was to break the statement in two:

Example:
```
    S temp = gorf();
    fun( &temp );
```

37. We have added support for:

    ```
    #pragma error "error message"
    ```

    Use the ANSI method because it is more portable and acceptable (Microsoft header files use the less portable #pragma when there is a perfectly fine, portable way to issue a message).

    The portable, acceptable method is:

    ```
    #error "error message"
    ```

38. We have added support for `__declspec(dllexport)`, `__declspec(dllimport)`, `__declspec(thread)`, and `__declspec(naked)` for Win32 (i.e., WinNT 3.5 and Win95) programs. Here are some examples:

    Example:
    ```
    __declspec(dllexport) int a;        // export 'a'
    variable
    __declspec(dllexport) int b()       // export 'b'
    function
    {
    }

    struct __declspec(dllexport) S {
        static int a;                   // export 'a'
    static member
        void b();                       // export 'b'
    member fn
    };
    ```

```
extern __declspec(dllimport) int a; // import ’a’
from a .DLL
extern __declspec(dllimport) int b();//import ’b’
from a .DLL

struct __declspec(dllimport) I {
    static int a;                   // import ’a’
static member
    void b();                       // import ’b’
member fn
};
```

39. The C++ compiler generates better error messages for in-class initializations and pure virtual functions.

    Example:
```
struct S {
    static int const a = 0;
    static int const b = 1;
    void foo() = 0;
    void bar() = 1;
    virtual void ack() = 0;
    virtual void sam() = 1;
};
```

40. We have fixed macro processing code so that the following program compiles correctly. The compiler was not treating "catch" as a keyword after the expansion of "catch_all".

    Example:
```
#define catch(n) catch(n &exception)
#define xall (...)
#define catch_all catch xall

main()
{
    try{
    }
    catch_all{
    }
}
```

41. We have fixed a problem where `#pragma code_seg` caused a page fault in the compiler when the code_seg was empty.

42. We have fixed a rare problem where a #include of a file that was previously included caused the primary source file to finish up if the CR/LF pair for the line that the #include was on, straddled the C++ compiler's internal buffering boundary.

43. We have added support for `#pragma message( "message text" ).` It outputs a message to stdout when encountered. It is used in Microsoft SDK header files to warn about directly including header files and obsolete files.

44. We have fixed #pragma code_seg/data_seg to properly set the class name of the new segment in the object file.

45. We have a fixed a problem with the -zm -d2 options that caused a compiler fault in some circumstances.

46. We have fixed default library records in .OBJ file so that user libraries are ahead of default compiler libraries in the linker search order.

47. We have fixed handling of intrinsic math functions so that the code generator will treat functions like sqrt as an operator.

48. We have added support for using OS-specific exception handling mechanisms for C++ exception handling during code generation. Enable it with the new -zo option.

49. `__stdcall` functions now have Microsoft Visual C/C++ compatible name mangling.

50. We have added a number of new functions to the C Library. These have been added to improve Microsoft compatibility.

```
dllmain (nt only)
libmain (nt only)
_access
_dos_commit
_dup
_ecvt
_fcvt
_fstat
_fstrdup
_gcvt
_itoa
_itoa
_locking
_lseek
_ltoa
_ltoa
_memicmp
_set_new_handler
_stat
_strdate
_strdup
_stricmp
_strlwr
_strnicmp
_strrev
_strtime
_strupr
_tolower
_toupper
__isascii
__iscsym
__iscsymf
```

51. In version 9.5, the linker used to include LIBFILE object files in reverse order (i.e., the last one listed was the first to be included). We have corrected this behaviour so that they are included in the order listed.

```
Directive           Old Order   New Order
-------------       ---------   ---------
FILE    obj_a           3           3
LIBFILE obj_b           2           1
LIBFILE obj_c           1           2
FILE    obj_d           4           4
```

In the above example, the object files will be included in the order indicated (LIBFILE object files are always included first).

Changes in 10.5 that may Require Recompilation

__stdcall If you use the __stdcall attribute within a program then you must re-compile the function definition and all callers of the __stdcall function.

__cdecl The __cdecl attribute is not heavily used in Win32 programming so the impact should be minimal but if you do use __cdecl within your own programs, a re-compilation will be necessary.

9.6 Major Differences from Version 10.0 LA

If you have .tgt files created with the Limited Availability or Beta Integrated Development Environment, when you load them, the target window may say "Obsolete Form: rename target type". If it does:

1. Select the target window by clicking in it,
2. Choose "rename target" from the target menu (a rename target dialog will appear),
3. Reselect the target type for this target (e.g., Win32 EXE), and
4. Select OK.

You should not continue to use .cfg files from the Limited Availability version of the compiler. Several new features have been added. Using the old files will cause problems.

The C++ compiler calling conventions have changed. Any program that passes a "data only" class or struct as a parameter, or returns a C++ object will need to be recompiled. We recommend that you recompile your application.

The C++ compiler now supports the use of the ***__export, __cdecl, __pascal, __stdcall*** and ***__syscall*** keyword on class definitions. These keywords will affect all members of the defined class.

9.7 Major Differences from WATCOM C9.5 /386

- The functionality of Watcom C/C++^{16} and Watcom C/C++^{32} is included in a single package.

- An Integrated Development Environment for Windows 3.x, Windows NT, Windows 95 and OS/2 PM is included.

- New, redesigned debugger with GUI interfaces for Windows 3.x, Windows NT, Windows 95 and OS/2 PM is included.

- The optimizer has been enhanced.

- C++ Class Browser

- New, redesigned user interface for the Profiler.

- New support for C and C++ precompiled header files.

- Windows resource editing tools are included:

  Dialog Editor
  Bitmap Editor
  Resource Editor
  Menu Editor
  String Editor
  Accelerator Editor

- Windows development tools are included:

  Dr. WATCOM (a post mortem debug utility)
  Spy (Windows message spy program)
  DDESpy
  Heap Walker
  Zoom

- On-line documentation is included.

- Microsoft Foundation Classes for 32-bit Windows applications (MFC 4.1) and 16-bit Windows 3.1 applications (MFC 2.52b) is included.

- Creation of FlashTek DOS extender applications is supported.

- Compiler executables have been created that run under all supported operating systems. They are located in the BINW directory.

Items No Longer Supported

- PenPoint development
- Debugging of Ergo OS/386 DOS extender applications
- DESQView remote debugging

Changes in 10.0 that may Require Recompilation

All C++ applications will require recompilation due to changes in the underlying object model. C applications should not require recompilation, but *you should recompile your application if you want to take full advantage new features in the debugger*. The changes to the C++ object model are:

- Virtual table layout changed (NULL entry at offset 0 removed)
- derived class packing adjusted to minimize padding bytes
- exception handling code is improved (incompatible with 9.5)
- name mangling for ’char’ reduced from two chars to one char