Lattice C 5.5

the C Compiler for your Atari ST/STE/TT Computer

Installation Guide

Requires:

- ✓ Atari 520ST upwards
  (1M+ memory required)
- ✓ Disk drive
  (2 floppies or hard disk advised)
- ✓ Mouse

HiSoft
High Quality Software

Lattice C 5.5 for the Atari ST/STE/TT

By HiSoft and Lattice, Inc.

Program:
designed and programmed by HiSoft and Lattice, Inc.

Manual:
written by Alex Kiernan and David Nutkins.

HiSoft
High Quality Software

Published by HiSoft

The Old School, Greenfield, Bedford MK45 5DE UK

Second Edition, June 1992

Table of Contents

Lattice C 5.5 Installation Guide

Welcome to Lattice C Version 5.5, one of the most powerful and flexible programming environments available for the Atari 680x0 range of computers.

The package contains 7 double-sided, double density disks but the system is usable on any machine with at least a megabyte of RAM and a double-sided disk drive.

The contents of the disks are arranged logically (we hope) so that you can get to the files you need quickly and easily.

Backups

The first thing you should do is make a backup. You have several ways of making a backup of the Lattice C 5.5 disks. You can use the disk copying function of the Desktop to duplicate the disks, or you can use one of the many disk copiers available. *Note however, you must ensure that the backup program preserves the volume name of the original disk as otherwise the installation program will complain that you have inserted the wrong disk.*

However you do it, *please make a backup* and then store the master disks in a safe place, away from moisture, extreme heat or cold, magnetic fields (televisions, telephones etc. give off radiation harmful to disks), strong light, coffee and, above all, children and dogs! If you damage your master disks we will charge you a handling fee for re-copying them.

Now, before we go any further... please read the READ.ME file from your backup of Disk 1. You can do this by double-clicking on it from the desktop. This file will detail any last minute changes that we may have made.

The Installation Program

The GEM-based installation program is designed to ease the building of various standard configurations for the Lattice C 5 system.

For hard disk owners, the installation program will copy the files that you need to your hard disk. If you are the type of person who doesn't like installation programs that write things to your hard disk, you can view the files that would be copied, and copy them yourself. The installation takes note of your hardware configuration and only copies files that could be of use to you.

By default, the installation program won't copy the compressed headers since you will normally get better performance with the standard ones. You also don't get the tools for using the GST format as most people don't need this. The installation program for hard disk users deliberately does not automatically install a ramdisk. This is because many users will already have their own preferred ramdisk. The recommended ramdisk size for 1Mb hard disk users is 100Kb, which is supplied ready to go on disk 3. We strongly recommend that you use your ramdisk for the compiler's intermediate (quad) files. These are set up using the QUAD environment variable. See your user manual addendum for more information.

For users of floppy disk based systems, the installation program will produce two floppies that you can use to develop your programs; one boot disk and one work disk. The boot disk will just contain program files; the work disk will be used for the header files, libraries and your own source and program files. If you have two drives you can keep the boot disk in drive A and the work disk in drive B. On single drive systems we recommend that you use the same scheme, letting the Atari's operating system prompt you to change disks; if you keep all your files on your work disk you shouldn't need to do this often!

If you are using a non-hard disk system and wish to use larger than normal capacity (e.g. 800K) floppy disks, you should format two floppies using your favourite extended formatter *prior* to running the installer. Use the volume names LC5BOOT and LC5WORK. If you intend to use standard floppies then the installation program will format these for you.

We strongly suggest that you start by using the setup recommended by the installation program until you are sufficiently familiar with the package to re-configure it to meet your unique requirements.

If you have a pre-Rainbow version of TOS, you *must* run the Atari program FOLDRnnn.PRG before running the installation program; otherwise the program will terminate with an out of memory error due to the '40 folder' bug in your version of TOS. If you have a floppy-based system just boot from your backup of disk 1. If you have a hard disk and don't have FOLDRnnn.PRG in your AUTO folder copy it from the AUTO folder of disk 1.

To run the installation program, double-click on LCINST.PRG from your backup of disk 1 in Drive A. Note that the installer expects to find its subsidiary files in the current directory.

The Disks

The disks and their contents are:

1. Integrated Tools

`bin\lc5.prg`	The editor and integrated shell
`bin\hisofted.inf`	The editor configuration file
`bin\default.prj`	Default project file for the integrated environment
`bin\lc1.lc`	English error messages
`bin\lc1.ttp`	Cut down first phase of the compiler
`bin\lc1b.ttp`	First phase of the compiler
`bin\lc2.ttp`	Compiler code generator
`bin\clink.ttp`	The Lattice linker
`bin\monstc.prg`	Debugger for 68000 machines
`bin\monttc.prg`	Debugger for 68030 machines
`lcinst.prg`	The installation program
`lcinst.rsc`	File used by the installation program
`lcinst.inf`	File used by the installation program
`lcdisks.dir`	File used by the installation program

2. WERCS & Command Line Tools

`bin\asm.ttp`	Lattice macro assembler
`bin\batcher.prg`	MS-DOS style shell
`bin\cpxbuild.ttp`	CPX builder
`bin\dercs.ttp`	Resource embedder
`bin\go.ttp`	Global optimiser
`bin\gstlib.ttp`	The GST format librarian

bin\lc2gst.ttp -------- Lattice to GST format converter
bin\lcompact.ttp ------ Header file compacter
bin\linkst.ttp -------- GST format linker
bin\omd.ttp ----------- Object module disassembler
bin\oml.ttp ----------- Object module librarian
bin\strip.ttp --------- Symbol removing utility

WERCS Program Files

wercs\wconvert.prg ---- Resource file name converter
wercs\wconvert.ttp ---- Resource file name converter
wercs\wercs.inf ------- WERCS settings file
wercs\wercs.lng ------- WERCS language description file
wercs\wercs.prg ------- WERCS
wercs\wercs.rsc ------- Resource file
wercs\wimage.prg ------ Image converter
wercs\wimage.rsc ------ Resource file

Non-C language WERCS Test programs

wercs\wtest\wrsc.mod
wercs\wtest\wtest.mod
wercs\wtest\wtest.pas
wercs\wtest\wtest.bas
wercs\wtest\wtest.hsp
wercs\wtest\wtest.s

3 Header files, Example, Extras

Un-compressed Header Files

The headers directory contains the pure ASCII versions of the header files and is useful when checking the prototypes or definitions of external objects.

headers\acc.h
headers\aes.h
headers\assert.h
headers\basepage.h
headers\conio.h
headers\cpx.h
headers\ctype.h
headers\dirent.h
headers\dos.h
headers\errno.h
headers\error.h
headers\ext.h
headers\fcntl.h
headers\fctype.h
headers\float.h
headers\fsm.h
headers\ftw.h
headers\gemextra.h
headers\gemlib.h
headers\gemout.h
headers\ieeefp.h
headears\io.h
headers\ios1.h
headers\limits.h
headers\linea.h
headers\locale.h
headers\memory.h
headers\m68881.h
headers\math.h
headers\osbind.h
headers\oserr.h
headers\portab.h
headers\process.h
headers\setjmp.h
headers\signal.h
headers\stdarg.h
headers\stddef.h
headers\stdio.h
headers\stdlib.h
headers\string.h
headers\strings.h
headers\taddr.h
headers\time.h
headers\tos.h
headers\utime.h
headers\unistd.h
headers\varargs.h
headers\vdi.h
headers\sys\file.h
headers\sys\except.h
headers\sys\param.h
headers\sys\time.h
headers\sys\dir.h
headers\sys\stat.h
headers\sys\types.h

Compressed Header Files

The headers in the h directory are the normal ASCII header files which have been processed by lcompact, resulting in space savings and time savings on floppy based systems. We do not recommend their use on hard disk systems.

Examples

examples\bell\cpx*.* - Source, resource and project files for bell.cpx

examples\bell\h*.* --- Header files used by the bell example

examples\bell\tsr*.* - Source and project files for belltsr.prg

examples\clock*.* ---- Desk accessory example

examples\cpx\master.* CPX resource file template

examples\gstlib*.* --- GST format librarian source and project files

examples\hc*.* ------- Example of use of the HiSoft C Interpreter library

examples\hc\lib*.* --- HiSoft C GEM Toolbox library and project files

examples\prog2.c ------ Lesson source file

examples\wtest*.* ---- WERCS example program

Source Files

The source code supplied in the src directory form assorted parts of the Lattice C runtime library. They are supplied so that the expert user may customise them as required.

src\c_assert.c ------- assert() failure code

src\c_cxovf.c -------- C library stack overflow code

src\c_cxovf.c -------- GEM library stack overflow code

src\c_iob.c ---------- ANSI level I/O blocks

src\c_main.c --------- _main() startup code

src\c_setargv.c ------ _setargv() argument parser

src\c_stack.c -------- Default stack size module

src\c_stub.c --------- _stub() undefined linker default

src\c\basepage.i ------ Include file used by c.s

src\c\c.s ------------- Initial program startup source

src\c\gemdos.i -------- Include file used by c.s

src\c\oserr.c --------- _OSERR messages

src\c\syserr.c -------- errno messages

src\c\sysvar.i -------- Include file used by c.s

src\g_aesaddr.c ------ AES control array

src\g_aesglob.c ------ AES control array

src\g_aesinti.c ------ AES control array

src\g_aesinto.c ------ AES control array

src\g_aespb.c -------- AES control array

src\g_vdictrl.c ------ VDI control array

src\g_vdiinti.c ------ VDI control array

src\g_vdiinto.c ------ VDI control array

src\g_vdipb.c -------- VDI control array

src\g_vdiptsi.c ------ VDI control array

src\g_vdiptso.c ------ VDI control array

src\m\cxferr.c -------- Floating point exception handler

src\m\cxfpe.c --------- Floating point exception handler

src\m\fpcfpcr.s ------- Floating point co-processor configuration

src\m\matherr.c ------- Maths error handler

HiSoft Ramdisk Tools

The ramdisk directory contains the reset proof ramdisk and its installation program - see the Lattice C 5 Tools, hramdsk chapter in Volume I - The User Manual.

ramdisk\hramdsk.prg --- HiSoft ramdisk program

ramdisk\raminst.prg --- HiSoft ramdisk installation program

ramdisk\raminst.rsc --- Resource file

Extras

aespath*.*	Program to fix a problem in 1.04 and 1.06 TOS
auto\belltsr.prg	Ready to run TSR for use with BELL.CPX
clock.acc	Ready to run example Desk Accessory
cpx\bell.cpx	Bell CPX. Requires DMA Sound hardware
extras\aes.i	Include file for use with Devpac and DERCS
extras\as68.syn	Synonyms for porting Atari MadMac files
extras\atari.s	Standard Atari equates file
extras\c.lnk	An example LinkST control file
extras\default.lnk	An example CLINKWITH file
extras\error.cmd	MicroEMACS file for LC error handling
extras\lc1.frg	LC1.LC in German
extras\lc1.usa	LC1.LC in American English
sounds*.*	Sound files for use with the Bell CPX example

4. Base-relative C and GEM Libraries

Disk 4 contains the startup code and standard C & GEM libraries for the base-relative (small data) memory models. The naming conventions for the start up code are described on Page 121 in the Second edition User Manual and Page 269 in the First edition User Manual. The start-up stubs that are suffixed with cpx are for producing Control Panel Extensions for use with Atari's Extended Control Panel.

5 Non-base-relative C & GEM Libraries

This disk contains the non base relative versions of the files on disk 4.

6 Software/Auto-detecting Maths Libraries

This disk contains the standard software-only and auto-detecting maths libraries for all memory models. The auto-detecting libraries (suffix ma) will automatically use a directly connected or I/O mapped coprocessor if present. The standard software-only version (suffix m) will not, but have the advantage of being considerably smaller than the auto-detecting libraries.

7 Co-processor Maths Libraries

This disk contains the maths libraries for directly-connected (suffix m8) and I/O-mapped (suffix mi) maths libraries. These require the corresponding hardware to be present but generate the fastest and most compact code for the particular system.

Hints for upgraders

If you've used a previous version of Lattice C 5 here are some points to note:

- The installation program may tell you that you don't have enough disk space to install the system if you try to install it 'on-top of' your existing installation. Either delete your version 5.0x files before running it or ignore the warning message from the installation.
- There are no GST libraries supplied on the disk. If you are using GST format - don't panic - the LC2GST program can now cope with entire libraries, so you can just convert the Lattice format libraries that you require.

- There are no longer separate LC.PRG and EDC.PRG programs. LC5.PRG is used for both. The editor now uses a single HISOFTED.INF file rather than the two separate files that existed in earlier versions.
- If you have existing make, batch or script files that use the linker's XADDSYM option change these to ADDSYM.

A Lattice C 5.5 Tutorial

Lesson 1 - Your First Program

We are now going to guide you through your first experience with Lattice C 5.5 (at least, we hope it's your first experience - you haven't been too impatient, have you?). First of all, you must install Lattice C 5.5 for your system. The method of doing this depends on your system (how much memory you have and whether you have a hard disk or not) - this is documented in the first part of this manual.

First of all, find the LC5.PRG program (wherever you have put it on installation) and, from the GEM desktop, double-click on the LC5.PRG icon to run the program. The following screen will appear:

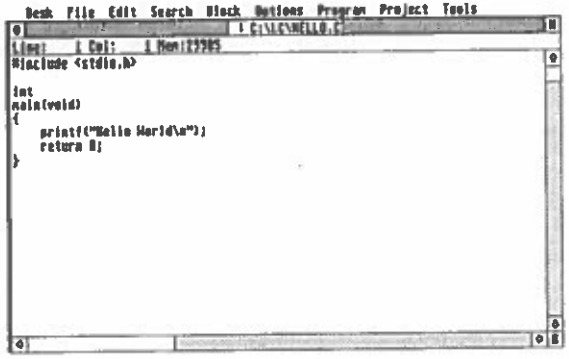
The LC5 Editor Opening Screen

Now type the following program using the keyboard in the normal way and pressing `Return` at the end of each line:

```
#include <stdio.h>

int
main(void)
{
   printf("Hello World\n");
   return 0;
}
```

Your screen should now look like this:



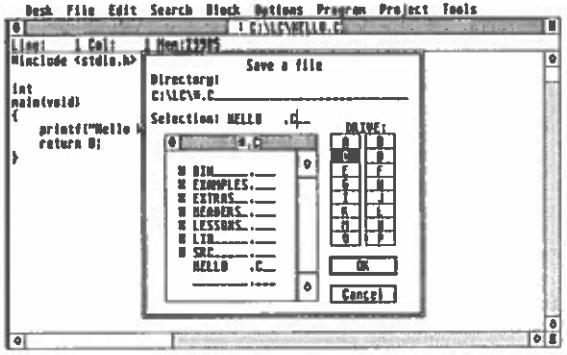
The Hello World *Program*

Before we compile this complicated program we really ought to save it to disk, in case anything goes wrong The best place to save your source files is the Lattice C 5.5 working disk or your hard disk if you have one. So press Alt-S or click on Save As... from the File menu and the file selector will appear.

The appearance of the file selector will depend on which one you have installed in your system. If your operating system is TOS 1.4 or greater, you have, automatically, an extended file selector with drive buttons such as that shown in the screen shot on the next page.

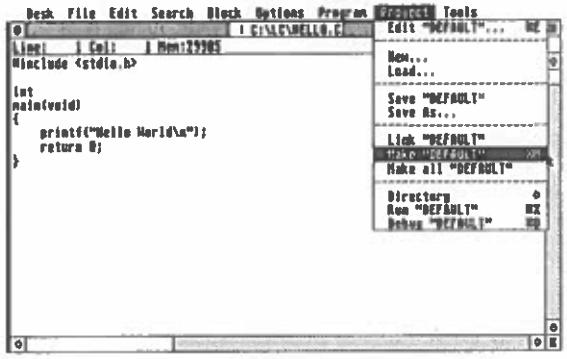
However, if you have an earlier version of the system ROMs, you will see, unless you have installed a new one, a much more primitive file selector, without drive buttons.

When the file selector appears, type in HELLO.C and click on the relevant drive button so that the box looks like this:



The File Selector

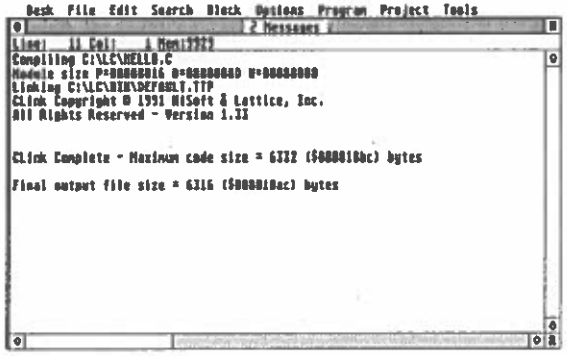
To build your program type Alt-M or select Make "DEFAULT" from the Project menu as shown below:



The Project *menu before* Make "DEFAULT"

The compilation and link will now proceed immediately and, assuming it is successful, you will see a window something like the one shown below.

If anything goes wrong, check that you have typed in the program correctly and that you have set everything else noted earlier in this Installation Guide to Lattice C 5.5 - otherwise the integrated compiler may not be set up.



The Compilation and Link of HELLO.C

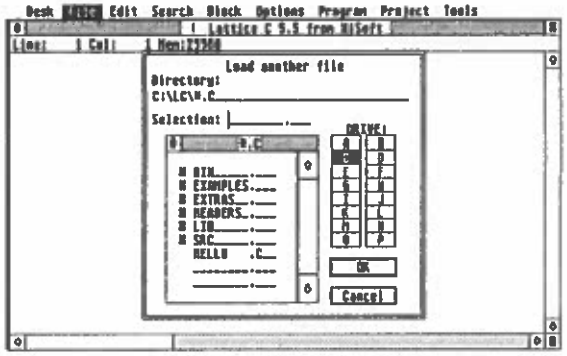
After this you will have a file called HELLO.TTP on your working disk - to run the program first change the top window so that your program (HELLO.C) is on top, thentype Alt-X or click on Run "DEFAULT" on the Project menu, then type Return when the box appears and you should see Hello World appear at the top of the screen, hit a key and you will be returned to the editor.

Running your First Program

Congratulations - you have created, compiled, linked and executed your first Lattice C 5.5 program!

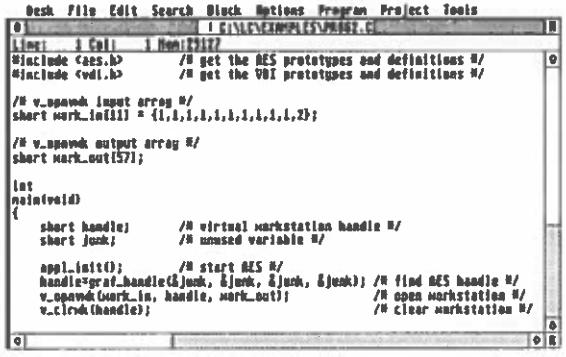
Lesson 2 - We all make mistakes

Run LC5.PRG, insert your working disk and type Alt-L or select Load... from the File menu.



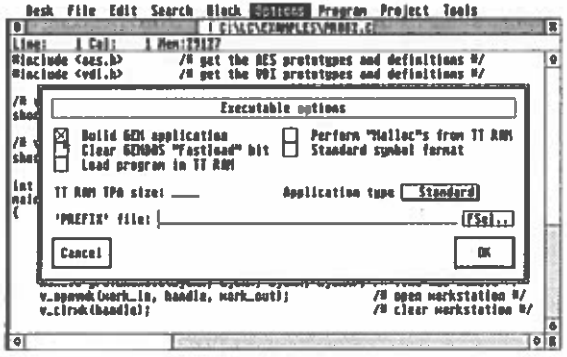
The Load File *Command*

Single-click on the EXAMPLES directory to open it and then double-click on PROG2.C to load the program into the editor.



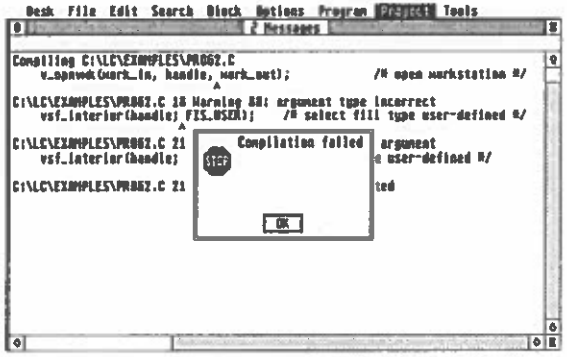
The PROG2.C *Program*

Now compile this program as you did in Lesson 1 but making sure that you select Build GEM application from the Executable options dialog (obtained by selecting Executable... from the Options menu).



Interactively Compiling with GEM

What happened? You should have seen a report something like this:

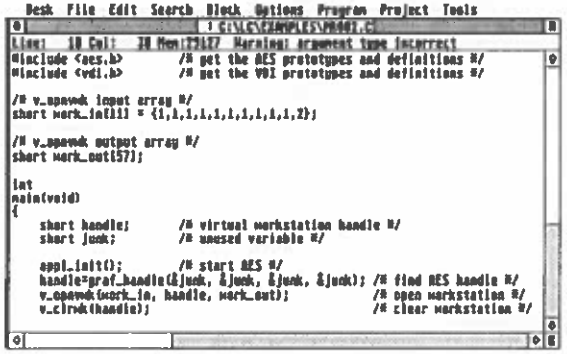


The first compilation of PROG2.C

The compiler has generated 1 warning and 2 errors.

Now get back to the editor with the PROG2.C program loaded. To achieve this simply hit return, or click on the OK box in the alert.

Now, how do we correct these errors... the cursor will already be positioned on the line in which the first error/warning occurred and the description of the problem will be in the top of the window.



Interactively Getting to the Error

Looking at the first erroneous line (`v_opnvwk(work_in, handle, work_out);`) it looks legal, but the compiler has given an argument type incorrect warning. This means that the type of one of the arguments in a function call was not what the compiler expected it to be. In fact, the problem here is that the second parameter in the v_opnvwk function has been declared in the header file as a pointer to a short and, as such, the *address* of `handle` (`&handle`) should be passed.

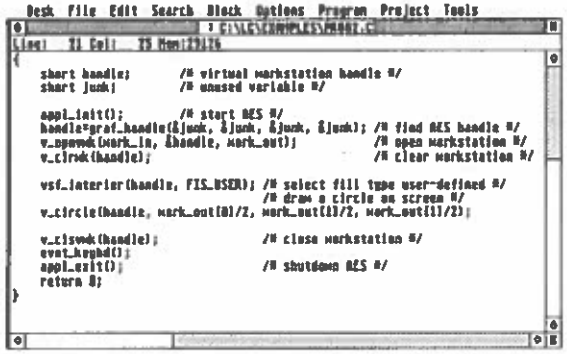
So position the cursor on the h of `handle` and type an ampersand character, &.

Now go to the next error by pressing Alt-J, this is in the line:

```
vsf_interior(handle; FIS_USER); /* select fill-type */
```

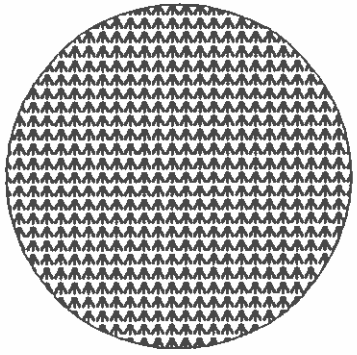
and the error is Invalid function argument. Well, it shouldn't take you too long to spot that the semi-colon between `handle` and `FIS_USER` should be a comma. Position the cursor over the space before the F of `FIS_USER` and hit Backspace followed by a comma - ,.

Now compile the program again, remembering to save it first. It should now compile and link successfully, to PROG2.PRG, and you can try running it.



The Corrected Program

The program draws a filled circle like this:



The PROG2.PRG program running

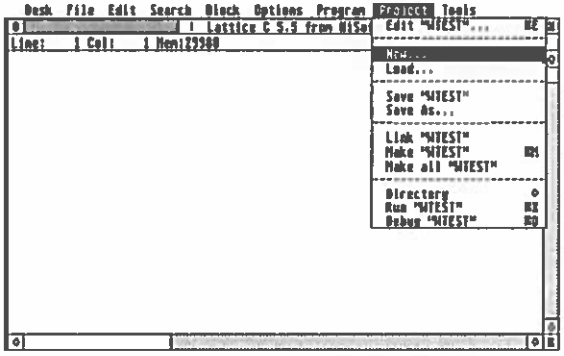
If you are wide awake you may have noticed that the compiler reported one warning and two errors (look back at the output on page 13) and that we only corrected the warning and one of the errors to obtain a successful compilation. This is because the second error was caused by the first error; the compiler became confused as to the meaning of the program and thought that a semi-colon was overdue. This is an example of a spurious error caused by a previous problem - always be on the look out for this type of error.

Lesson 3 - Optionally yours

The program we are going to build is an extended example of using the GEM system and uses structures created with the WERCS resource editor - see the chapter WERCS, The Resource Editor in Volume I for more details.

Our concern here is to show you how to set up the project which we will use to compile it, in addition to using some useful compiler options; these are used to let you modify the way the compiler behaves when compiling your program.

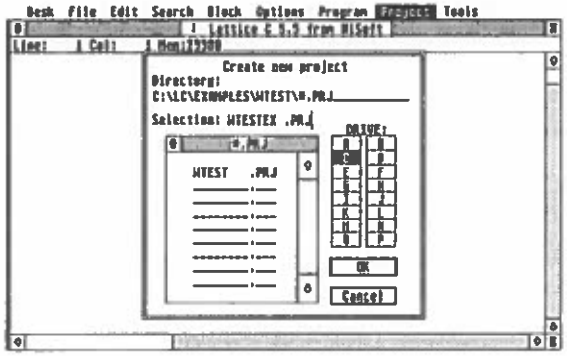
Run LC5.PRG, insert your working disk and then select New... from the Project menu.



Preparing to create a new project

When the file selector appears, locate the WTEST directory inside the EXAMPLES directory on your work disk and then type in the name WTESTEX.PRJ, this is the name of the file we are going to use as our *project file*. A project file is a file which contains all of the compiler options and source files which are need for a particular project.

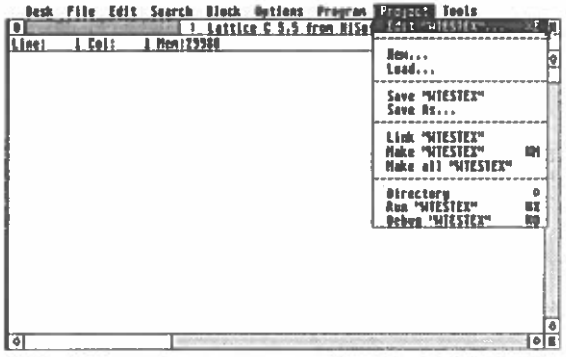
Note that there will already be a .PRJ file in the EXAMPLES\WTEST directory, do not select this file! The file is a ready-made version of the project file which we are going to build, but supplied for the benefit of those users not working through the tutorial.



Creating a new project

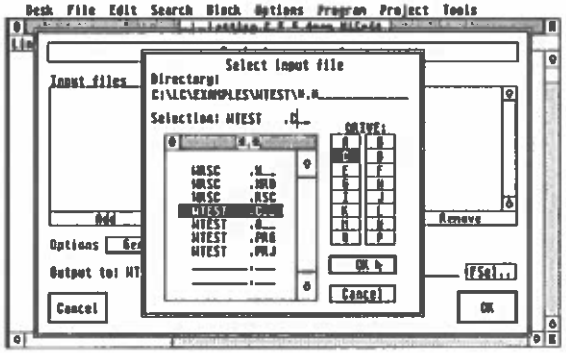
Having told the integrated compiler that we are starting on a new project, we must now add the source files and their dependencies which go to make up our final project.

Type Alt-E or select Edit "WTESTEX"... from the Project menu to open the Project management dialog.



Preparing to edit the project

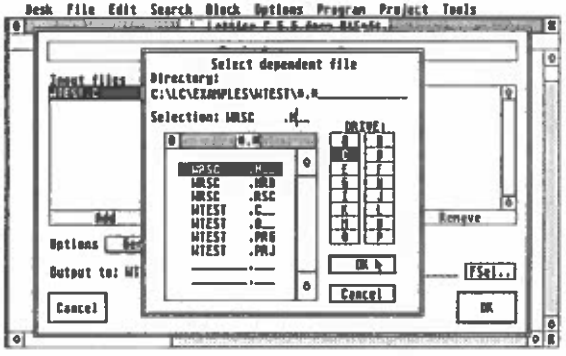
For the WTEST example program we have only one source file, so to add this click on the Input files - Add button (the first button from the top on the left hand side). You should see a file selector like the one below:



Adding the input file WTEST.C *to the project*

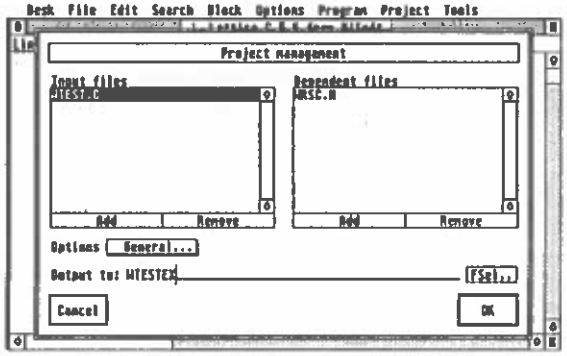
Within this file selector double-click on the WTEST.C source file (or select it and then select OK).

Having done this the WTEST.C source file will be selected. We now need to add the user file which WTEST.C `#includes`, WRSC.H (a *dependent* file). To do this click on the Dependent files - Add button. Again a file selector is displayed, this time select the WRSC.H file, as shown.



Adding the dependent file WRSC.H *to* WTEST.C

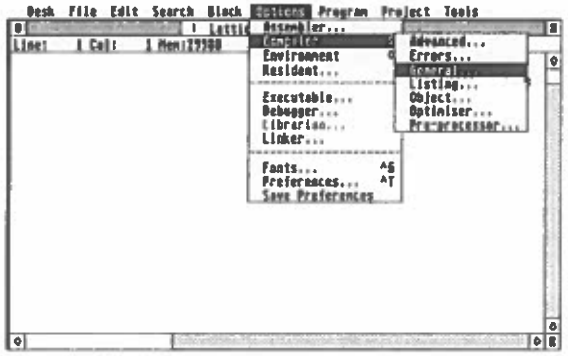
Having accomplished these steps, the Project management dialog should now look like this:



The completed project, WTESTEX.PRJ

Now click on OK to accept all the changes we have made.

In addition to setting up the project, we want to select a number of compiler options which are going to be used for this project. Select the Options - Compiler - General... option as shown below.



Selecting the General *compiler options*

Within the Compiler options - General box we are going to select the following options:

Disable stack checking

this option tells the compiler not to generate any inline code in your program that will check that there is sufficient stack space during your program's execution. You would normally only use this option when you had a completely finished program that had been fully tested and that you wished to be of a minimum size and to run at maximum speed.

Enforce function prototypes

function prototypes were introduced into the C language by the recent ANSI standard and are used to tell the compiler exactly how a function should be called, thus making it easier for you. Lattice C 5.5 checks for a function prototype and, if this option is used, will warn you if a prototype is absent.

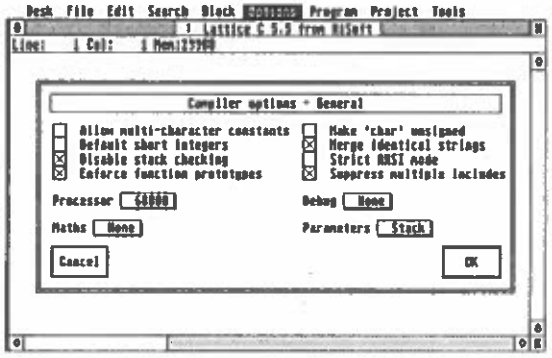
Merge identical strings

this keeps the program smaller by only keeping one copy of any identical constant strings.

Suppress multiple includes

suppresses multiple `#includes` of the same file. If a second `#include` of the same file is encountered, the directive is simply ignored, since the compiler does not need to even open the file when it encounters it a second time. This can give improved compiler performance.

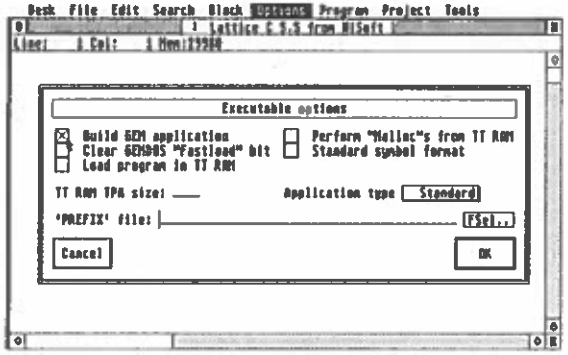
The Compiler options - General box should now look like this:



Setting the General *compiler options*

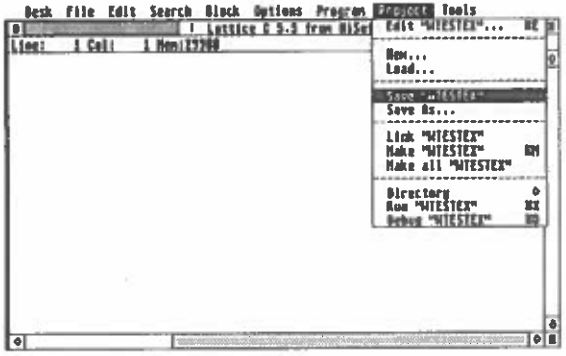
Now click on OK to accept the changes we have made.

Because this is a GEM program we also need to tell the compiler to build a GEM application; we do this, as in lesson two, using the Executable options dialog:



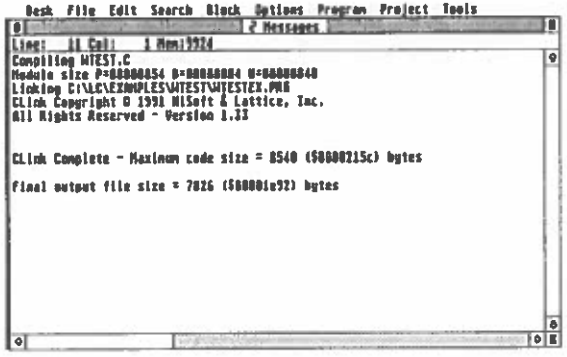
Setting a GEM type program

Having finished setting up our project we should now save the project file for future use, so select the Save "WTESTEX" option from the Project menu:



Saving the completed project file

We are now ready to build the project in the normal way, by selecting Make "WTESTEX" from the Project menu; you should see a compilation report something like this:



A successful 'Make'

The WTESTEX.PRG program will now be in the WTEST directory of your Work disk, and we can run it using the Run "WTESTEX" from the Project menu; the running program looks like:

WTESTEX.PRG *running*

That completes our brief but, we hope, useful introduction to using the Lattice C 5.5 system. We now encourage you to get on and use the package, referring to this and the other volumes as and when you need to.