Commands reference

Every source file that you wish to debug must be compiled with the **Let's C** compiler, using the **-VCSD** option to the **cc** command.

You may link in objects that are not compiled with **-VCSD**. In the evaluation window, the global variables defined in these objects will be visible, but **csd** will have no knowledge of the internal variables of objects not compiled with **-VCSD**.

Invoking csd

To start **csd**, type **csd** followed by the name of the program you want to debug. If you invoke **csd** on a program and receive the error message:

```
out of space
```

while reading source, you need to use the large model source debugger, **lcsd**. **lcsd** is the version of the debugger that has a large data segment. To invoke the large model debugger, type **lcsd** followed by the name of the program you want to debug.

For Tandy 2000 users, type **tcsd** to invoke the debugger. For any other Tandy PC compatible, invoke the debugger with the **csd** command.

To debug the program **infl**, type:

```
csd infl
```

When you invoke the debugger, it will print in the history window:

```
C Source Language Debugger Version 1.1
Copyright 1984-1988 by Mark Williams Co., Lake Bluff, IL
Reading tables...
Adding source:  infl.c
Loading infl...
```

The line

```
Reading tables...
```

shows that **csd** is reading the debug information for the program, and

```
Loading infl...
```

means that it is loading the **.exe** program file. The line

```
Adding source:  infl.c
```

means that **csd** is adding the source file used to build the program. **Adding source:** followed by a file name appears once for each source file **csd** adds.

Typing **csd** at the prompt without arguments produces the following reminder:

```
usage: CSD -G -D -O[L][M] -Hhelppath -Ssourcepath -T file[.exe] args
```

The information following "CSD" refers to command line options, which are discussed below. **file** is the name of a **.exe** file you wish to debug. The **.exe** suffix is optional. **args** is the remainder of the command line to be supplied to your program (as if it were running without **csd**). For example, when you type

```
csd factor 234
```

the number **234** is the argument passed to the program **factor**.

The **csd** options must *precede* the file name, so that they are not confused with the arguments to your program. The **csd** options may be entered in any order.

Options

The following describes the options to the **csd** command.

-O[L][M]

User program is:

-O small model object module format
-L large model object module format
-M small model Mark Williams format

Ordinarily, **csd** determines the format of the **.exe** file you are using by the number of relocations to be done at load time (this is part of the **.exe** header).

0 relocations means MWC format
1 relocation means small OMF
>1 relocations means large OMF

If a small model program contains a **.s** file with absolute segment references (the **@** operator), these references must be relocated, causing the above rule to be violated. Note that the Mark Williams format was produced by **Let's C** in versions earlier than 4.0. It is no longer used by **Let's C**. These options allow debugging of such **.exe** files.

-D*nnn* Change *data segment* size. If you are using a version of **csd.exe** called **lcsd** that has a large data segment, the **-D** option lets you specify the size of **csd**'s workspace. *nnn* is a decimal number of kilobytes; the default is 128. This workspace holds disk buffers, user expressions, and the history. It is allocated in addition to the fixed memory used to hold your program's symbol table. If this option does not appear in the usage reminder, you are using a **csd.exe** with a small data segment; if this is the case, the data segment is 64 kilobytes, the maximum possible.

-H Set the *help path*. This option tells **csd** where to find its help files. By default, **csd** looks for help files in the current directory, then in the directories named by the environmental variables **PATH** and **LIBPATH**.

The help path must be a complete MS-DOS directory specification. It should terminate in a back slash (\), the only exception being a drive specification (e.g., **C:**).

-G The *graphics* option. Use this when running **csd** on a color system that is not in 80x25 character mode.

-I*nn* The reset *interrupt* option. **csd** ordinarily uses MS-DOS interrupt 3 to set tracepoints in your program. However, if the program you are debugging already uses interrupt 3, you may want **csd** to set tracepoints using a different interrupt. The **-I** option tells **csd** to use interrupt *nn*, where *nn* is a decimal number.

-S *source path*

-T *source path*

These options tell **csd** where to find your C source files.

When you compile a C source file with the **-VCSD** option, the file name that you type (which

may include a directory specification) is saved within the object modules and the executable file. For example, if you typed

```
cc -VCSD \source\example.c
```

the executable program would remember the name of its source file not as **example.c**, but as **\source\example.c**. This name will be displayed in the reverse-video line between the source and evaluation windows.

The **-S** option tells **csd** to look for C source files in the named *source path*. **csd** will prepend the given directory path to each source file name in the symbol table. **-S** is useful if you compiled the program using the file names with no directory specification, and you want to use the debugger from another directory without recompiling. For example, if you compiled the program with

```
cc -vcsd program.c
```

and invoked **csd** with the command

```
csd -Sutility\program
```

csd will look for the source file **utility\program.c**.

The **-T** option also tells **csd** to look for C source files in the named *source path*, and it strips the existing directory information from the saved source file names. **csd** will strip any directory path names from the source file names it finds in the symbol table, then prepend the given path before reading in the files. **-T** is useful if you compiled the program giving path names that are no longer relevant and should be disregarded. For example, if you compiled the program using the command line

```
cc -vcsd \src\program.c
```

and invoked **csd** with

```
csd -T\utility\program
```

csd will look for a source file called **\utility\program.c** and ignore the leading directory name it found in the symbol table.

This feature is helpful in situations where you would rather not rebuild an executable file or change directories. For simplicity's sake, it is suggested that you run **csd** from the same directory where you compiled the program being debugged, or specify complete pathnames for the source files given to the **cc** command.

Exiting csd

To exit the debugger once you have started debugging, press the **<Shift-F1>** key. If you are currently typing a line in the evaluation window, type **<ctrl-U>** before exiting.

Getting help

Whenever the debugger is waiting for your input, you can type the **<F6>** key and the debugger will display the master help screen. This help information describes all the major functions of the debugger and the key that invokes each function. From this screen, you can choose other help screens that describe individual functions in detail.

After you have read the help screen, type **<F6>** again to return to the debugger.

The following help screens are provided:

```
Cursor movement
General
Trace
Run
Evaluation
Find
History
Insert/Delete
Select
```

The help screens are detailed enough so that many users do not need to refer to this manual once they get started. Section 6, *Help screens*, contains a copy of every help screen.

Windows

csd uses four windows: the *source window*, the *evaluation window*, the *program window*, and the *history window*.

The *source window* displays your program's source code. By moving the cursor within this window, you can tell **csd** which portion of code to evaluate, where to set tracepoints, and which stack frame to use.

The *evaluation window* is where **csd** accepts C expressions and variables that you type on the keyboard. When an expression is typed into this window, **csd** evaluates it and prints the result back in the evaluation window.

The *program window* is the area in memory where a program normally writes its output.

The *history window* is where **csd** records the events of your debugging session. Each time **csd** stops your program at a tracepoint, the traced statement or expression is written onto the history window. This is **csd**'s way of logging statements that it encounters during a debugging session.

Program window

To see the program window, type **<F7>**. **csd** normally jumps to the program window while your program is running, so you can see the output that your program has generated. This screen appears just as it would if your program were executed without **csd**.

You can choose to display a screen or window other than the program window while your program is running. To do so, use the **<F2>** key. Keep in mind that the program's output will go to the selected screen when you run the program. To erase the output, press **<F7>**, followed by the key for the selected screen or window.

Source window

The source window contains the source code of the program you are debugging. The cursor appears in this window when you first invoke **csd**. To return to this window from another screen or window, type the **<F8>** key.

If your program is more than one screen long (about 19 lines), not all of your program can be displayed at one time. To scroll through your program one line at a time, use the <↑> and <↓> arrow keys. The <PgDn> and <PgUp> keys move through your source program one **page** (or screen full of text) at a time. Finally, the <Home> and <End> keys jump to the beginning or the end, respectively, of your entire source file.

If you want to look at a line that contains a specific string, use the **<F1>** key. This provides a powerful search feature with which you can find a specific string and locate the next statement to be traced. The **<F1>** key is discussed in more detail later in this section.

To execute your program, press the **<F4>** key, then one of the following:

<F3>	Execute to the next tracepoint.
<return>	Execute one line of code.
<↓>	Same as <return>, but function calls are treated as one statement.
<←>	Execute to the end of the current function.
<Home>	Reload and initialize program.
<End>	Execute to the end of the program; do not stop at tracepoints.

If you have been moving through your source window and have lost track of what the next executable statement is, press the **<Shift-F8>** key. This will automatically position the cursor at the next executable statement.

When **csd** stops at a tracepoint in your source program, the cursor is positioned at the beginning of the traced source line, which is the next line to be executed.

csd displays both the source window and the evaluation window at the same time. The boundary between them is a reverse-video line that names the source file being displayed. If your program uses more than one source file, the name will change as you scroll from one file to another. **csd** displays the source files in the order in which you entered them on the **cc** command line.

You can change the relative sizes of the evaluation and source windows with the **<F2>** key, which will be described later in this section.

Evaluation window

The evaluation window is beneath the source window, under the reverse-video line. To jump to the evaluation window, press the **<F9>** key. This window is where you can type expressions that you want **csd** to evaluate. For example, if the variable **index** is an **int** that is in the current scope of the program and has the value '1', you can display its value by typing

```
index
```

into the evaluation window. **csd** will evaluate **index** and print the following:

```
index :: 1
```

Values are displayed in their natural format except for **structs** and unions, which are displayed as a list of hexadecimal bytes. For large structures, **csd** will display as much as will fit on the current line. For purposes of tracing, large structures are, of course, evaluated in their entirety. Strings are printed within quotation marks, and characters are enclosed within apostrophes.

You can type any legal C expression into the evaluation window. **csd** will check the legality of the expression as you type it. If you make an error in syntax or if a variable name is typed incorrectly, **csd** will give you an error message. To delete a character that is mistyped, use the backspace key. Typing **<ctrl-U>** deletes an entire line of text.

csd can evaluate expressions that use the following elements:

operators
constants (except '#' defined constants)
variables (in the current scope)
functions

In addition, the following special casts allow you to display information in a useful form:

oct octal
hex hexadecimal
str character strings

For example, if you issue the command

```
csd factor 12 22 32
```

then execute one statement by typing **<F4>** and **<return>**. Now you can enter the evaluation window, and type the following statements:

```
argv[1]
(str)argv[1]
(oct)argv[1]
```

csd will fill the evaluation window as follows:

```
argv[1] :: 0x5321
(str)argv[1] :: "12"
(oct)argv[1] :: 051441
```

Note that pointers are automatically displayed in hexadecimal. Keep in mind that **argv[1]** contains a pointer allocated by the run-time startup library routine and might have a different value for your system.

csd cannot evaluate expressions that contain C keywords, braces, labels, or semicolons. One exception to this rule is the keyword **sizeof**: it can be used in the evaluation window. Also, preprocessor information is not available in the evaluation window: **csd** does not recognize '#' defined symbols.

Expressions are evaluated as soon as they are typed in, and also whenever your program encounters a tracepoint, provided they are in the current scope.

Another powerful feature of **csd** is its ability to set tracepoints on expressions entered in the evaluation window. This causes **csd** to halt execution of your program if the value of the traced expression changes. To do this, **csd** enters single-step mode to evaluate all of the expressions in the evaluation window before each line of source code is executed.

To set a tracepoint on an expression in the evaluation window, switch to the evaluation window; then type the expression to be evaluated, but instead of typing **<return>**, type **<F3>**. The expression will shift into high-intensity characters to show that it is being traced. From now on, when you run the program in trace mode, execution will stop whenever the value of the traced expression changes.

When you type in expressions that contain variables, the variables must be in the current scope. This means that the variables that you are entering must be either external or declared by the function within which the cursor is positioned. Expressions that refer to several different scopes can appear in the evaluation window at the same time; however, **csd** cannot evaluate all of them at the same time. **csd** stops execution when it encounters a tracepoint and checks the expressions within the evaluation window; it evaluates only the expressions that are within the current scope.

Note that evaluation window expressions that produce side effects, such as assignments or **printf** statements, should be used with caution and deleted after use. Remember, too, that if you are tracing an expression, each expression in the window is evaluated after *each* statement in the program. Because the debugger needs to halt the program after each line to evaluate any traced expressions, execution is slowed considerably.

You can manipulate the evaluation window by using the following keys:

csd C source debugger

Up <↑>	Move up to previous expression
Down <↓>	Move down to next expression
Page Up <PgUp>	Move to previous page of expressions
Page Down <PgDn>	Move to next page of expressions
Begin <Home>	Move to beginning of evaluation window
End <End>	Move to end of evaluation window
Find <F1>	Move to pattern specified
Delete <Del>	Delete portions of the evaluation window
Insert <Ins>	Add expression at cursor

Please note the following:

1. Preprocessor information is not available in **csd**.
2. Literal strings may be used only as function arguments; you may only reference objects in your own data space. You can affect one assignment, as follows:

```
cp=strcpy(malloc(4), "cat")
```

3. Structure initializors, for example **{0, NULL, 135}**, are not expressions. Structures must be initialized one member at a time.

History screen

The history window is called by striking the <F10> key. This screen displays a log of events that occur while debugging, including every line of code or expression that was executed or traced while it had a tracepoint set.

You can manipulate the history window using the same keys as above, except **Insert**.

Up <↑>	Move up to previous line
Down <↓>	Move down to next line
Page Up <PgUp>	Move to previous page of lines
Page Down <PgDn>	Move to the next page of lines
Begin <Home>	Move to beginning of history window
End <End>	Move to end of history window
Find <F1>	Move to pattern specified
Delete <Del>	Delete portions of history window

Because the history window and the other I/O activity done by **csd** share the same buffers in memory, the contents of the history window are deleted automatically by **csd** as memory gets full.

You can redirect history window entries to a printer by using the **<F2>** command with the '**L**' option. Keep in mind that a program with many traced statements and expressions will generate large amounts of output.

Command keys

csd supports a number of different keystrokes for any given command. The following table lists each **csd** function and cursor movement command and the corresponding keys to invoke the command. The following sections describe the cursor movement commands.

FUNCTION	DESCRIPTION	COMMAND
Find	Find a string of text	<F1> or <esc>1 or <ctrl-S> or <esc>S or <ctrl-R> or <esc>R
Select	Select an option	<F2> or <esc>2
Trace	Trace an expression	<F3> or <esc>3
Run	Run the program	<F4> or <esc>4
Cancel	Cancel the last command	<F5> or <esc>5 or <F5>
Help	Display a help screen	<F6> or <esc>6 or <F6>
Program	Display program window	<F7> or <esc>7
Source	Display source window	<F8> or <esc>8
Evaluation	Enter evaluation window	<F9> or <esc>9
History	Display history window	<F10> or <esc>0
Up	Move cursor up	<↑> or <ctrl-P>
Down	Move cursor down	<↓> or <ctrl-N>
Out	Move to calling function	<←> or <ctrl-B>
In	Undo an **Out**	<→> or <ctrl-F>
Exit	Exit **csd**	<Shift-F1> or <ctrl-X> <ctrl-C>
Current	Return to current line	<Shift-F8> or <ctrl-X>X
Page Up	Move cursor up a page	<PgUp> or <esc>V
Page Down	Move cursor down a page	<PgDn> or <ctrl-V>
Delete	Delete a line	<Del> or <ctrl-K>
Insert	Insert a line	<Ins> or <ctrl-O>
Beginning	Beginning of source	<Home> or <esc><
End	End of source	<End> or <esc>>

Begin <Home>

The <Home> key can be used by itself to move the cursor or with another key to modify that key's action.

<Home>
Move the cursor to the beginning of the source, evaluation, or history windows. This command positions the cursor on the first line of the material in the screen or window.

<Del> <Home>
In the evaluation window or history window, remove all material from the beginning to the current cursor position.

<F1> <Home>
In the source window, the evaluation window, or the history window, search for a pattern from the current cursor location back toward the *beginning* of the window. In the case of the source window, *all* source files are searched.

<F4> <Home>
Reload and reinitialize the target program.

End <End>

The <End> key can be used by itself to move the cursor, or with another key to modify that key's action.

csd C source debugger

<End> Move the cursor to the last line of the source window, the evaluation window, or the history window.

<Del> <End>
Erase all lines from the current cursor position to the end of the window or screen. This works in the evaluation and history windows only.

<F1> <End>
In the source window, the evaluation window, or the history window, search for a pattern from the current cursor location through the end of the window. In the case of the source window, *all* source files are searched.

<F4> <End>
Execute the target program to the end, without stopping at tracepoints. All tracepoints, however, are logged into the history window.

Up <↑>

The <↑> key can be used by itself to move the cursor, or with another key to modify that key's action.

<↑> Move the cursor to the previous line in the source window, the evaluation window, or the history window.

<Del> <↑>
Erase the current line and move the cursor to the previous line. This works in the evaluation and history windows only.

<F1> <↑>
In the source, evaluation, or history windows, search for a pattern from the current cursor location through the beginning of the window. In the case of the source window, only the current source file is searched, beginning at the current cursor position and moving toward the first line of the current source module.

<F2> <↑>
In the source or evaluation window, increase the size of the evaluation window by one line and decrease the size of the source window by one line.

Down <↓>

The <↓> key can be used either by itself to move the cursor, or with another key to modify that key's action.

<↓> Move to the next line in the source window, the evaluation window, or the history window.

<↓> <Del>
In the evaluation window or the history window, remove the current line and move the cursor to the next line.

<F1> <↓>
In the source window, the evaluation window, or the history window, search for a pattern from the current cursor location through the end of the window. In the source window, search only the current source file.

<F2> <↓>
In the source or evaluation windows, decrease the size of the evaluation window by one line and increase the size of the source window by one line.

<F4> <↓>
Execute one line of code. A function call is treated as one single line of code. This is in contrast to the sequence **<F4> <return>**, which executes a single statement, even if the next

statement is within a called function.

Out <←>

The <←> key is used in the source window to change the current scope of a variable. When your program stops at a tracepoint, the active scope is that of the function being executed. Using the <←> key activates the scope of the function that called the current function. Then you can examine or change variables defined in the scope of the calling routine as well. When the scope is changed, the cursor is moved to the line of code that called the current function. Repeatedly pressing the <←> key will bring you to the outermost stack frame.

In <→>

When used in the source window, the <→> key undoes the effect of the <←> key. If you have not typed the <←> key since the last tracepoint, the <→> key has no effect.

Page Up <PgUp>

When used in the source window, the evaluation window, or the history window, the <PgUp> key scrolls the current window one page (or screen full of text) toward the beginning. If the current page of the source window is less than one page from the beginning of the file, the cursor will be moved to the beginning of the file.

If the cursor is positioned in the source window near the beginning of a source file, the <PgUp> key will move the cursor into the previous file, should there be one.

Page Down <PgDn>

In the source window, the evaluation window, or the history window, pressing the <PgDn> key moves the cursor one page toward the end. If the cursor is less than one page from the end, it will be moved to the end of the file.

If the cursor is in the source window and positioned near the end of a source file, the <PgDn> key will move the cursor into the next file, if there is one.

Find <F1>

The **<F1>** key searches for a specific pattern. For this reason, it is also called the *find* key.

The action of the **<F1>** key can be modified by a number of different keys, as follows.

<F1> <F3>
In the source and evaluation windows, find the next statement that has a tracepoint set on it.

<F1> <Home>
In the evaluation window and the history window, search toward the beginning. In the source window, search toward the beginning of the window and examine all files.

<F1> <↑>
In the evaluation window and the history window, search toward the beginning. In the source window, search toward the beginning but search only the current source file.

<F1> <End>
In the evaluation window and the history window, search toward the end. In the source window, search toward the end and examine all files.

<F1> <↓>
In the evaluation window and the history window, search toward the end. In the source window, search toward the end, but search only the current source file.

csd C source debugger

The string that **<F1>** seeks is a *pattern* of characters. These characters can be ordinary alphanumeric characters, or a mixture of alphanumeric characters with *wildcard* characters that modify how the search is conducted.

The simplest pattern is a set of non-special characters, which is matched literally.

The following special characters can be used to specify powerful patterns:

^	Match only the beginning of a line
$	Match only the end of a line
?	Match any one character
*	Match any number of characters
\	Escape character: use it to search for a wildcard literally
[*abc*]	Match any one of *a*, *b* or *c*
[*a-m*]	Match any of the letters *a* through *m*

To match a line in the source, evaluation, or history windows that ends in a ';', type the sequence

<F1>;$<return>

This says to find a pattern that has a semicolon followed by the end of a line.

Keep in mind that **<F1>** will search for the strings exactly as they appear on the screen. For example, to find the line

```
a :: 11
```

where **11** is the value supplied by **csd**, type:

<F1>1$<return>

If you want to search for any of the special characters of the patterns, precede that character by a backslash '\'. For example, to find the line

```
c = a*b;
```

type:

```
a\*b
```

Note that if you do not type a pattern after the **<F1>** key, **csd** will search for the last pattern used, if there is one.

If you start an **<F1>** command and wish to abandon it or restart it, press the **<F5>** key. Typing **<ctrl-U>** restarts the find pattern.

Insert <Ins>

The <Ins> command is used only in the evaluation window. It opens a blank line so you can enter a new expression above the one entered in the current line.

Delete <Del>

The <Del> key can be used in the evaluation window or the history window to remove or "kill" text. Its action can be modified by other keys, as follows:

<Del> <↑>
Kill the current line of text, and reposition the cursor on the previous line.

<Del> <↓>
Kill the current line of text, and reposition the cursor on the following line.

<Del> <Home>
Kill all text from the beginning of the file to the cursor.

<Del> <End>
Kill all text from the cursor to the end of the file.

To cancel a kill command before it is executed, type **<F5>**.

You should periodically use **<Del>** to purge the history window of information you no longer need. If you do not, it will take up space that could be used to hold source lines in memory, and slow down **csd**.

Run <F4>

The **<F4>** key, which can be used in any window, tells **csd** to execute your program. Its operation is modified by the following keys.

<F4> <F3>
Execute the program up to the next tracepoint. The tracepoint can be either in the source window, in which case execution stops just before execution of the traced line, or in the evaluation window, in which case **csd** stops if the value of the traced expression changes. The cursor is moved to the statement or expression that bears the tracepoint, unless it is in a different window.

<F4> <return>
Execute one line of code. If that line is a function call, execution stops on the first line of that function.

<F4> <↓>
Execute one line of code, as with the **<return>** modifier discussed above, but treat function calls as a single line of code.

<F4> <→>
Execute to the end of the current function.

<F4> <Home>
Reload and restart the program being debugged.

<F4> <End>
Execute the program to its end, without stopping at any traced statements or expressions. Traced statements and expressions will be logged onto the history screen.

To cancel **<F4>,** press the **<F5>** key.

Trace <F3>

The **<F3>** sets a tracepoint on the line on which the cursor is currently positioned. Traced lines are written in high-intensity characters. To turn a tracepoint off, simply move the cursor back to the line being traced and press **<F3>** again. Tracepoints can be set either in the source window or in the evaluation window.

When a tracepoint is set on a line of code in the source window, execution stops immediately *before* that line of code. Only executable lines of code can be traced; statements such as comments and declarations cannot be traced. Note, too, that because the compiler optimizes unused and some common code out of existence, some things that appear to be executable actually cannot be executed. If you try to set a tracepoint on a line that has no executable code, the debugger will give you the error message:

```
not executable statement
```

When a tracepoint is set on an expression in the evaluation window, execution of the program stops when the value of the traced expression changes.

Tracing an evaluated expression causes **csd** to single-step through the entire program. This is done so **csd** can recognize the change in the variable as soon as possible. Thus, tracing an expression in the evaluation window noticeably slows the execution of the program being debugged. The execution will also be noticeably slower if a large number of statements are executed if you Run to the next line, or Run to the end of the function.

Select <F2>

The **<F2>**, or "select", key controls the way **csd** presents its screens. How it functions is controlled by a number of modifying keys, as follows.

<F2> <↑>
Move the boundary between the source and evaluation windows up by one line.

<F2> <↓>
Move the boundary between the source and evaluation windows down by one line.

<F2> <F7>
Display the program's output on the program window. This is the default.

<F2> <F8>
Display the program's output in the source window. The source window will shift to display the code currently being executed. Note that if no tracepoints are set, execution may be too fast to follow.

<F2> <F9>
Display the program's output in the evaluation window. The variables in the evaluation window will change as the program executes. Note that if no tracepoints are set, execution may be too fast to follow.

<F2> <F10>
Display the program's output on the history window.

<F2> L Redirect to the printer all material normally logged onto the history window. Pressing this combination of keys again turns off this feature.

To cancel an **<F2>** command, press the **<F5>** key.