# CSi-Locate™

# User's Guide

Version 2.0
June 1996

## Copyright and Trademark Information



**United States Software Corporation**

7175 NW Evergreen Parkway ♦ Suite 100

Hillsboro, OR 97124

USA: 800-356-7097

TEL: 503-844-6614 ♦ FAX: 503-844-6480

E-mail: info@ussw.com ♦ www.ussw.com

# Table of Contents

# Manual Organization

This CSi-Locate User's Guide contains the following chapters:

### *1. Locating with CSi-Locate*

This chapter describes the location process, explains features of and how to use CSi-Locate, and gives troubleshooting guidance.

### *2.16-bit Real-Mode Applications*

Look in this chapter to find worked-out examples of locating in RAM (only) and in ROM (and RAM) 16-bit real-mode applications. There is significant overlap in chapters 2–4; reading one of these three chapters will probably suffice as you get started.

### *3. 16-bit Protected-Mode Applications*

Look in this chapter to find a worked-out example of locating in RAM (only) a 16-bit protected-mode application. There is significant overlap among chapters 2–4, so reading one of these three chapters will probably suffice as you get started.

### *4. 32-bit Protected-Mode Applications*

Look in this chapter to find a worked-out example of locating in RAM (only) a 32-bit protected-mode application. There is significant overlap among chapters 2–4, so reading one of these three chapters will probably suffice as you get started.

### *5. Command Reference*

The CSi-Locate .CMD command file helps make locating applications easy and fast. This chapter describes command files in detail and also provides a detailed explanation of each of the 38 commands that can go into them.

### *Appendix A. Error and Warning Messages*

This appendix explains the error and warning messages that CSi-Locate generates.

### *Appendix B. ROMming Protected-Mode Applications*

This appendix provides general guidance on how to ROM protected-mode applications.

***Step-by-step installation DOS-based instructions***

1. Create a new subdirectory on the disk drive where you want to install CSi-Locate. For explanatory purposes here and throughout this manual, we will assume that the subdirectory you create is CSILOC.

   ```
   mkdir csiloc
   ```

2. Make it the current directory.

   ```
   cd csiloc
   ```

3. Place the installation disk in the floppy drive from which you want to install CSi-Locate.

4. Invoke the install program with the following command:

   ***x:*INSTALL *x:***

   where ***x*** is the name of the floppy drive into which you have inserted disk 1 (both ***x*s** are required in the command).

See the file FILES.DOC (an ASCII file that can be found in the directory where you installed CSi-Locate) for a list of files installed during installation. Another file that you will want to look at is the file READ.ME, which discusses known problems and restrictions, along with changes made to the software after this manual went to press. This file can also be found in the directory where you installed CSi-Locate.

**Typographical Conventions Used in This Manual**

| | |
|---|---|
| FILENAME.EXT | Directories, filenames, and filename parts are set in small capitals |
| **LOCATE** | Locator commands are in all capital letters and set in bold Helvetica |
| **debug** | Code and command examples are set in bold Courier |

# Locating with CSi-Locate

# 1

CSi-Locate locates real-mode, protected-mode, and mixed-mode applications by performing commands that you give to it in a sequential command file. When combined with one of the popular compiler-linker combinations that it supports (listed on page 4), the CSi-Mon monitor, and the Soft-Scope debugger, it becomes part of a complete solution for embedded-system development.

Since many programmers are new to the location process, we provide a general description of this important process starting on the next page. Then we provide an overview of the capabilities of CSi-Locate in the next section of this chapter, highlighting its distinctive characteristics. The next topic is tool chains that CSi-Locate supports, followed by a general discussion of the input CSi-Locate requires and the output it produces. A .CM conversion map file is a sort of listing file always generated by the locator and that file type is discussed next. The final section of this chapter provides troubleshooting assistance.

## Table of Contents

# The Location Process

***Linking/locating***

The simple story of the linking-locating process is that linkers order and link program segments together, while locators assign addresses to them. Linkers in fact do more than that and locators similarly have historically taken on more duties than just address assignment. We will assume that you have a general familiarity with what goes on in the linking process, but we will take a closer look at the location process here.

***Native vs. embedded development***

Native-development environments today shield the user almost entirely from the locating process because native applications are practically always relocatable. Embedded-system developers, however, need to know about locating because their applications must usually be absolutely located. The primary difference between relocatable and absolutely located applications is that the latter have fixed addresses.

***Location units***

When building an absolutely located application, you work with the following three basic structures:

Segments — Real-mode and 16-bit protected-mode segments are blocks of code or data that can range in size from 0 bytes to 64KB. Protected-mode 32-bit segments can be as large as 4GB. Of the three possible 80X86-family type of segments (namely, data, code, and system), applications programmers create and work primarily with the data and code segments. The locator can be used to set up system segments (e.g., those containing tables for the GDT, IDT, LDT, TSS, and the various trap and interrupt gates), as well as define code and data segments at location time that don't exist until then, but are used by the user's application.

Classes — A collection of segments, either grouped together according to user specifications or classed together by the linker to organize modules. For example, all of the segments in one class may contain initialized data, and all the segments in another class may contain initialization code or code written in assembly language.

Groups — Groups are also collections of segments, but each of the segments in a group has the same segment base. So, the entire group must be within the segment-size limit defined by the processor mode, which is 64KB for real mode and 16-bit protected mode and 4GB for 32-bit protected mode.

Consider the following when locating these structures in memory:

***Know where your RAM and ROM are***

You need to know the starting addresses and sizes of RAM and ROM chips on your target board.

Segments that don't have load-image data (for example, the Stack and BSS segments) can be located, but their contents will be undefined because no load-image data is present. If you use **LOCATE** to specify that the data images of such segments be written to the target, those data images will be set to zero. Also, zero-length segments are not placed in the output .ABS, .HEX, or .BIN files.

***Default location order***

All segments not explicitly located using commands are located in the order defined by the input MAP file or the order in which they are created with the locator command **CREATE**.

Typically you will want to locate code in ROM and data in RAM. The examples worked out in chapter 2–4 should help you get a feel for using CSi-Locate and for the location process generally.

***What CSi-Locate does***

CSi-Locate orders and absolutely locates in memory the segments, classes, and groups that constitute 80x86-family (aka *Intel Architecture*) executable programs, creating code that can be loaded into RAM using the Soft-Scope debugger or burned into ROM to create an embedded application.

***Compilers supported***

CSi-Locate supports applications built with tool chains (that is, assemblers, C/C++ compilers, and linkers) from the following vendors, for both DOS and Microsoft Windows executables:

- Borland
- Microsoft
- Watcom

***Sets up protected-mode strcutures***

It can construct 16- and 32-bit protected-mode CPU structures, including the GDT, IDT, LDT, gates, page tables, and TSSs, and it supports multiple-mode or mixed-mode applications.

***Supports initialized RAM data***

If your application has initialized data that you want to reside in RAM, the locator can compress the segments containing it for storage into ROM, and then at boot-up time your application's startup code can uncompress the data and copy it into RAM by using macros supplied with the locator. These uncompress-and-copy macros are named **`raminit_16r`** (for 16-bit real mode), **`raminit_16p`** (16-bit protected), and **`raminit_32P`** (32-bit protected), and can be found in the file CSILOC\CSILOC.INC.

***Other support macros***

Here is a list of other support macros (and what they do) that can be found in the file CSILOC.INC just mentioned:

```
def_alias          Define alias segments.
def_gate           Define CALL/TASK/TRAP/Interrupt gates.
def_noentrygate    Define gate w/out default entry point.
def_init           Define ram init table segment.
def_tbl            Define GDT, IDT, LDT, and page tables.
def_tss            Define TSS segments.
sup_defseg         Support macro for def_xxx macros.
sup_init16         Support for raminit_xxx macros.
sup_init32         Support for raminit_xxx macros.
```

***Command-file driven***

To make CSi-Locate easy to use, all commands and options are read from a sequential command file (see pp. 52–55 for a detailed discussion of the .CMD command file), and the user controls the ultimate output format.

***Possible output formats***

Possible output formats of your located application are as follows:

- Intel OMF86 absolute
- Intel OMF286 absolute
- Intel OMF386 absolute
- Intel extended 86 hex
- Intel 32-bit hex
- Binary

Default extensions for output files are as follows:

| | |
|---|---|
| absolute | .ABS |
| hex | .HEX |
| binary | .BIN |

A type of listing file, called a conversion map file (with CM extension), is always generated by CSi-Locate; see following pp. 6–7 for an annotated example .CM file.

***Required input files***

CSi-Locate requires access to the linker-generated executable and map files. These files provide symbolics, load image, segment/class/group names, and fixup information that CSi-Locate uses to prepare your output files.

For ease of use, files are distinguished by their extension, which makes it possible to use the same filename prefix for the executable file, the map file, and the .CMD command file. For example, each of our sample programs, which can be found in the subdirectories of CSILOC\SAMP, uses the following files:

**csamp.exe**
**csamp.map**
**csamp.cmd**

By default, CSi-Locate assumes that the executable and map files have the same path and filename prefix that the command file has. You can, however, specify different paths or names for those files by means of the I/O locator commands **EXEC** and **MAP**, described in chapter 5, pp. 70 and 80, respectively.

# Sample Conversion Map File

***...gure 1.1. Conversion map file***

```
Conversion map listing file     Aug  23, 1995  09:14:05PM

CSi-Locate embedded application locator, version 2.01
(c) Concurrent Sciences, inc. 1993-1996
Serial No.  MANUAL
```

Header with the date, time, CSi-Locate version number, copyright, and serial-number information

```
Processing command file.
[1] //
[2] // Microsoft C/C++ 386 32-bit flat model sample
[3] // This file is set up to work with the default
[4] // configuration of Concurrent Sciences' CSi-Mon
[5] // debug monitor and to operate out of RAM
[6] //
[7]
[8] cpu 386      // Target is 80386
[9] absolute     // Produce output file in OMF386
[10] debug       // Generate symbolics

Reading executable file.

[11] locate .text :: 8000p
[12] create systss      // Create segments to hold GDT,
[13] create sysgdt       // GDT and IDT tables and
[14] create sysidt       // initial TSS
[15] gdt sysgdt[3..64] :: reserve      // CSi-Mon needs
                                       //slots 3 to 64
[16] idt sysidt[0..40] :: reserve      // Reserve for
                                       //Intel CPU
[17] tss386 systss :: cs:eip=_boot ss:esp=start_tos
                       // Set up initial TSS values
[18] gdt sysgdt :: *    // Enter all segments in GDT
[19] fixup selector start_data = group flat_d
[20] fixup far32 start_stack = start_tos
[21] fixup far32 start_code = _main
[22] fixup far32 start_init = raminit
```

Command file listing

```
CSi-Locate input/output files:

Command script:    bigc.cmd
Conversion map:    bigc.cm
Input executable:  bigc.exe  (32-bit Windows executable)
Input map:         bigc.map
Absolute output:   bigc.abs
Hex format output: (Not generated)
Binary output:     (Not generated)
```

Input/Output file names

```
Protected-mode segment map:
Logical         Physical   Length     Name      Class Group   Memory

0208:00000000   00007000P  000000c0   INIT_TEX  CODE  FLAT_C  ROM
    :00001000   00008000P  000025fc   .TEXT     CODE  FLAT_C  ROM
0210:0000       4000b000P  0000b0c8   .BSS      DATA  FLAT_D  ROM
    :00010000   00017000P  00000063   .RDATA    DATA  FLAT_D  ROM
    :00013000   0001a000P  00004000   STACK     DATA  FLAT_D  ROM
    :00017000   0001e000P  00000039   .EDATA    DATA  FLAT_D  ROM
0218:0000       00021000P  00068      SYSTSS                  ROM
```

Located segment map (partial)

```
Initial GDT:  SYSGDT[0..67]
GDT[0]    0000  Empty      00000000L Lim=00000H DPL=0 gbp av
          .
          .
GDT[3]      0018  Reserved
GDT[4]      0020  Reserved
          .
          .
          .
```

Descriptor table information (partial)

```
Initial TSS286:  SYSTSS

gdt SYSGDT[65] 0208 Avail 286 TSS  00004390L Lim=0002bH DPL=0 gbP av

EAX=0000   EBX=0000   ECX=0000   EDX=0000
ESI=0000   EDI=0000   EBP=0000
DS=0000   ES=0000  FS=0000  GS=0000
LDTR=0000 LINK=0000
CS:EIP=0218:000E
SS:ESP=0220:4040
SS0:SEP0=0000:0000   SS1:ESP1=0000:0000   SS2:ESP2=0000:0000
EFL=0000 [ ac vm rf nt IOPL=0 of df if tf sf zf af pf cf ]
...
```

TSS information (partial)

```
Translating debug symbolics

Debug Symbolics Translation Complete:
    Modules. . . . . 23
    Procedures . . . . 42
    Public Symbols . . 145
    Source Lines . . . 1175
    Total Symbols. . . 263
    Types. . . . . 308
```

Public symbol information summary

***Figure 1.1. Conversion map file***

***CSi-Locate produces files for use with Soft-Scope source-level debuggers***

In addition to the output files mentioned above, CSi-Locate can create symbolic .BUG files for real-mode applications and can add symbolic information to protected-mode absolute files that make it possible for you to debug your application using the Soft-Scope family of debuggers (see the **DEBUG** command, p. XX).


If you want to debug your application using Soft-Scope, it must be prepared in one of the ways described in the tools sections at the ends of chapters 2–4.

Soft-Scope debuggers were designed specifically for embedded systems, and provide many of the features often associated with emulators and logic analyzers. For example, Soft-Scope for the CSi-Mon monitor can set software breakpoints, set hardware breakpoints on all 80x86 processors 386 or greater, stop target execution, view memory while the target is running, display a software trace, and log data to a file.

In addition, using a Tektronix logic analyzer and CSi-Connect, you can view your application's symbols on the logic analyzer and correlate the analyzer's hardware trace and powerful breakpoint capabilities with Soft-Scope's easy to use source-code interface.

Soft-Scope is also available in versions that support the Intel iSDM and iM-III monitors.

CSi-Locate does not support debuggers other than Soft-Scope.

| 16-Bit Real-Mode Applications | | | |
|---|---|---|---|
| Compiler | Assembler | Linker | Memory Model(s) for Applications |
| Borland C/C++ | Borland TASM | Borland TLINK | Segmented |
| Microsoft C/C++ | Microsoft MASM | Microsoft LINK | Segmented |
| Watcom 86 C | Watcom WASM | Watcom WLINK | Segmented |

*Table 1.1. 16-bit Real-Mode Applications*

| 16-Bit Protected-Mode Applications | | | |
|---|---|---|---|
| Compiler | Assembler | Linker | Memory Model for Applications |
| Borland C/C++ | Borland TASM | Borland TLINK | Segmented |
| Microsoft C/C++ | Microsoft MASM | Microsoft LINK | Segmented |
| Watcom 86 C | Watcom WASM | Watcom WLINK | Segmented |

*Table 1.2. 16-bit Protected-Mode Applications*

| 32-Bit Protected-Mode Applications | | | | |
|---|---|---|---|---|
| Compiler | Assembler | Linker | Memory Model(s) for RAM (only) Applications | Memory Model(s) for ROMmed Applications |
| Borland 32-bit C/C++ | Borland TASM | Borland TLINK | Flat or Segmented | Flat (with paging |
| Microsoft 32-bit C/C++ | Microsoft MASM | Microsoft 32-bit Executable Linker | Flat or Segmented | Flat (with paging) |
| Watcom 386 C/C++ | Watcom WASM | Watcom WLINK | Flat or Segmented | Flat or Segmented |

*Table 1.3. 32-bit Protected-Mode Applications*

### *1. Locked-together segments*

Locked-together segments are segments that share the same selector. They occur only in non-Windows executable files, and under the following conditions:

- When the first segment's length is less than 16 bytes long and the second segment is not paragraph aligned.
- When the first segment's length is zero, the second segment will share the same selector even if the second segment is paragraph aligned.

In the CSi-Locate .CM conversion map file (see annotated example on pp. 6–7) locked-together segments list the selector for the first segment in the group, but the other segments locked to the first do not show their selectors.

Here are three ways to avoid this problem:

1. pad the first segment with enough bytes to make it longer than 16 bytes;
2. if the first segment's length is greater than zero, paragraph align the second segment;
3. create a dummy public symbol, as in the following example (note that the type LABEL requires no allocated memory), in the segment(s) you want to locate separately:

```
public XYZ
XYZ label word
```

### *2. Group problems in 16-bit applications with the Microsoft and Borland compilers*

When the Microsoft and Borland compilers generates 16-bit non-Windows executable files, they create insufficient group information for segments that are not referenced by the application. CSi-Locate places these segments in a group with the preceding segment. Since the application doesn't reference these segments, this is usually not a problem.

However, it is possible that under some circumstances grouping segments in this manner would be undesirable.

To resolve this problem create a reference to the segments mistakenly included. Do this by putting the following statement in the assembly startup code inside some random segment:

```
DW seg_name
```

where,

*seg_name* is the segment mistakenly included in the wrong group

***3. Compressed data larger than 64KB in 16-bit applications***

The **INIT16P, INIT16R,** and **INIT32P** commands compress data and store it in ROM for initialization. For 16-bit applications, if the compressed data are larger than 64k you need to create two ROM segments and use the **INIT16P** or **INIT16R** command to initialize both of them.

***4. Assembly-language segments located incorrectly***

If a segment defined in assembly language gets located a few bytes beyond where you specify with the **LOCATE** command in a .CMD file, define the segment as paragraph aligned in your assembly code to prevent such dislocation.

***5. Truncated segments***

If the locator undesirably truncates a segment or segments that have been padded with zero-bytes (or padded in some other way), use the **INTEGRITY** command (see p. 77) to direct the locator to preserve those padded bytes and not truncate those segments.

This is the only line of text on this page.

# 16-Bit Real-Mode Applications

# 2

This chapter covers preparing and locating 16-bit real-mode applications. There are three examples of such applications supplied with CSi-Locate and they can be found in the following subdirectories:

CSILOC\SAMP\BCC16R (Borland)
CSILOC\SAMP\MSC16R (Microsoft)
CSILOC\SAMP\WCC16R (Watcom)

This chapter begins by discussing the example in CSILOC\SAMP\BCC16R, which is built to be run in RAM only. Then it disscusses a more complicated 16-bit real-mode ROMmed example. These examples will illustrate how you can prepare and locate your own 16-bit real-mode application. Tools for use with this type of application are discussed in the last three sections of the chapter.

## Table of Contents

# 16-bit Real-Mode Example to be Run out of RAM

***Step 1— Compile and link using Borland tools. For other tool chains, see the Tools section on pp. 24–25 of this chapter***

We have used batch files to create the sample programs included in all the CSILOC\SAMP subdirectories. For the sake of illustration in this chapter, we will discuss the sample program found in SAMP\BCC16R, which is composed of one assembly-language file (B16RBCC.ASM) and two C files (CMAIN.C and CUTILS.C), and built with Borland tools. This application is designed to be downloaded by Soft-Scope and run from RAM (see p. 8 for more on Soft-Scope). The following is the file BUILD.BAT that assembles, compiles, links, and locates the application.

```
: Batch file to create csamp.abs
: To use type 'build'
:
: Tools versions used:
:    Borland Turbo Assembler v4.1
:    Borland C++ v4.5
:    Borland Turbo Link v7.00
:    Concurrent Sciences CSi-Locate v2.02
:
tasm /Zi /mx b16rbcc.asm
bcc -ml -f -r- -v -O- -c cmain.c
bcc -ml -f -r- -v -O- -c cutils.c
tlink /m /s /v /Twe @csamp.flt
csiloc csamp.cmd
```

Our primary focus in this chapter will be on the command file CSAMP.CMD on p. 16, but before we scrutinize it, we will discuss the map file that plays an important role in the location process.

***Step 2—Examine your application's MAP file for class, segment, and group information***

When you build your application, you need to direct your linker to generate a map file because CSi-Locate requires one as one of its inputs. The map file on the next page was generated as part of the build process defined above. The detailed segment map allows you to see the classes, segments, and groups that you need to locate, and what their relationships are to each other. Some linkers, e.g., Microsoft's, don't show group information, which can prevent you from seeing locked-together segments (see pp. 10–11 for more on this type of problem).

In this case, since RAMINIT is the first segment among the segments ordered by the linker, if this is the only segment you locate, all the remaining segments will be located as a consequence of their relationship with RAMINIT.

*Fig. 2.1. MAP file*
***CSAMP.MAP***
***(modified format)***

```
Start         Length Name                   Class

0001:0000     0010H  RAMINIT                INIT
0002:0000     0096H  INIT_TEXT              CODE
0002:00A0     0403H  RESET_TEXT             CODE
0002:04A3     002FH  CMAIN_TEXT             CODE
0002:04D2     0199H  CUTILS_TEXT            CODE
0003:0000     0048H  _DATA                  DATA
0003:0050     4000H  STACK                  STACK
0003:4050     000CH  _BSS                   BSS

Detailed map of segments

0001:0000 0010 C=INIT    S=RAMINIT        G=(none)  M=B16RBCC.ASM ACBP=68
0002:0000 0096 C=CODE    S=INIT_TEXT      G=(none)  M=B16RBCC.ASM ACBP=48
0002:00A0 0403 C=CODE    S=RESET_TEXT     G=(none)  M=B16RBCC.ASM ACBP=68
0002:04A3 002F C=CODE    S=CMAIN_TEXT     G=(none)  M=CMAIN.C     ACBP=28
0002:04D2 0199 C=CODE    S=CUTILS_TEXT    G=(none)  M=CUTILS.C    ACBP=28
0003:0000 0000 C=DATA    S=_DATA          G=DGROUP  M=B16RBCC.ASM ACBP=48
0003:0000 0000 C=DATA    S=_DATA          G=DGROUP  M=CMAIN.C     ACBP=48
0003:0000 0048 C=DATA    S=_DATA          G=DGROUP  M=CUTILS.C    ACBP=48
0003:0050 4000 C=STACK   S=STACK          G=DGROUP  M=B16RBCC.ASM ACBP=74
0003:4050 0004 C=BSS     S=_BSS           G=DGROUP  M=CMAIN.C     ACBP=48
0003:4054 0008 C=BSS     S=_BSS           G=DGROUP  M=CUTILS.C    ACBP=48


Address          Publics by Name              Address          Publics by Value

0002:04A0 idle   HARDWARE_RESET               0001:0000 idle   RAMINIT_INIT
0001:0000 idle   RAMINIT_INIT                 0002:0000 idle   START_CODE
0002:0000 idle   START_CODE                   0002:0004 idle   START_DATA
0002:0004 idle   START_DATA                   0002:0006 idle   START_STACK
0002:000A idle   START_INIT                   0002:000A idle   START_INIT
0002:0006 idle   START_STACK                  0002:000E idle   _BOOT
0003:4050 idle   START_TOS                    0002:04A0 idle   HARDWARE_RESET
0002:000E idle   _BOOT                        0002:04A3        _MAIN
0002:04D2        _C_DATA                      0002:04D2        _C_DATA
0002:061B        _DELAY                       0002:05DB        _DISPLAY_LIGHTS
0002:05DB        _DISPLAY_LIGHTS              0002:061B        _DELAY
0003:4054 idle   _LIGHTS                      0003:4050 idle   _PATTERN
0002:04A3        _MAIN                        0003:4050 idle   START_TOS
0003:4050 idle   _PATTERN                     0003:4054 idle   _LIGHTS
```

***Step 3—Create a command file***

Use an ASCII text editor to create a.CMD command file with the commands that direct CSi-Locate to locate your application. Just below is a (slightly) modified version of the file SAMP\BCC16R\CSAMP.CMD, which is about the simplest possible example of a command file. We will begin to explain it in detail in numbered paragraphs immediately following it. The full story of the locator's 38 commands and how to construct command files with them occurs in chapter 5.

***Example .CMD file***

```
//
// Borland C++ real mode sample command file
// This file is set up to work with the default
// configuration of Concurrent Sciences CSi-Mon
// debug monitor
//

debug                  // Create debug file
absolute               // Create OMF86 absolute file
print
cpu 386               // Target is a 386 running in
                      // real mode
locate raminit :: 4000p
                      // Locate the first segment at
                      // 4000 physical
```

***Explanation of example .CMD file***

1. Note that comments begin with double slashes and end with a carriage return.
2. The order in which commands occur in the command file is significant. It is helpful to think of CSi-Locate as though it were an interpreter that processed each command as soon as it read it in. To help you get started with ordering commands, we provide general guidelines for command ordering in .CMD files on pp. 54–55. In general, place I/O commands first (the first four commands in the above example are I/O commands) and location and table-construction commands last.
3. The **DEBUG** command directs the locator to output symbolic debugging information that is essential for the Soft-Scope

*Explanation of example .CMD file (continued)*

debugger. For real-mode applications, CSi-Locate puts that information in a .BUG file that has the same name as the .CMD file. For protected-mode applications, the symbolics are placed in the .ABS absolute output file.

4. The **ABSOLUTE** command tells the locator to output an absolute file in Intel OMF format. The precise OMF type is determined by the application's mode (real or protected) and the target CPU, which is made known to the locator by means of the **CPU** command. See page 60 for further details on how the locator determines the OMF type.

5. The **PRINT** command directs the locator to print symbolic information about pubic symbols to the .CM conversion map file, which is automatically produced by the locator. An annotated example .CM file is given on pp. 6–7.

6. The **CPU** command specifies the target CPU to the locator. A list of possible values in the 80x86 family to be used with this command is given in table 5.5 on p. 67. The value used in the command partly determines the OMF type of absolute files (see 4. above) and is used in other ways by the locator to produce optimal output.

7. The **LOCATE** command plays perhaps the most crucial role among all the locator commands, for obvious reasons. This command tells the locator to absolutely locate one or more segments at a given address, which in turn may cause other segments (possibly all the segments in an application, as in the example here) to be located. This kind of ripple location effect results from the linker's relative ordering of segments: once the first segment in an ordered collection of segments is located, all subsequent segments as determined by the linking process fall into place. The map file produced by the linker can be used to see how segments are ordered.

   The following command achieves the exact same effect:

   ```
   locate * :: 4000P
   ```

   In this context, the * (asterisk) refers to all segments that have not yet been located, and the linker's ordering of segments is the default ordering used by the locator here too.

# 16-bit Real-Mode Example to be Run out of RAM

***Step 4—Invoke CSi-Locate***

To create a located application, use the following syntax at the DOS prompt:

CSILOC *filename*

where *filename* is the command file (with default extension CMD) that contains the locator commands, as in the following example:

```
csiloc csamp
```

If CSi-Locate locates your application as specified without error, the output file(s) that you request with the commands **ABSOLUTE**, **BINARY**, **HEX**, and **DEBUG** (the last creates a .BUG file for real-mode applications) are created, and a conversion-map file *FILENAME*.CM, which is a sort of locator listing file, is also produced. If the locator encounters any errors while trying to locate your application, the only output file is the .CM file, which shows all warnings and error messages generated during the location attempt.

Parts of the .CM file produced for our example 16-bit real-mode application are given below, with annotations. A more complete .CM is illustrated on pp. 6–7.

[In CSAMP.CM] Segment map that shows located segment addresses

```
Real-mode segment map:
   Logical     Physical  Length   Name          Class Group  Memory
-------------------------------------------------------------------
0400:0000      00004000P 00010    RAMINIT       INIT         ROM
0401:0000      00004010P 00096    INIT_TEXT     CODE         ROM
    :00a0      000040b0P 00403    RESET_TEXT    CODE         ROM
    :04a3      000044b3P 0002f    CMAIN_TEXT    CODE         ROM
    :04d2      000044e2P 00199    CUTILS_TEXT   CODE         ROM
0468:0000      00004680P 00048    _DATA         DATA  DGROUP ROM
    :0050      000046d0P 04000    STACK         STACK DGROUP ROM
    :4050      000086d0P 0000c    _BSS          BSS   DGROUP ROM
```

This should say RAM instead of ROM.

# 16-bit Real-Mode Example to be Run out of RAM

```
public RAMINIT_INIT . . . . . 0400:0000
public START_CODE . . . . . 0401:0000
public START_DATA . . . . . 0401:0004
public START_STACK. . . . . 0401:0006
public START_INIT . . . . . 0401:000a
public _BOOT. . . . . 0401:000e
public HARDWARE_RESET . . . . . 0401:04a0
public _MAIN. . . . . 0401:04a3
public _C_DATA. . . . . 0401:04d2
public _DISPLAY_LIGHTS. . . . . 0401:05db
public _DELAY . . . . . 0401:061b
public START_TOS. . . . . 0468:4050
public _PATTERN . . . . . 0468:4050
public _LIGHTS. . . . . 0468:4054
```

[In CSAMP.CM]
Public symbol information generated by PRINT command

```
Debug Symbolics Translation Complete:
    Modules. . . . . 3
    Procedures . . . . 6
    Public Symbols . . 14
    Source Lines . . . 212
    Total Symbols. . . 47
    Types. . . . . 56
```

[In CSAMP.CM]
Data on symbolics

```
Conversion complete, No errors or warnings.
```

[In CSAMP.CM]
Final statement

# 16-bit Real-Mode ROMmed Example

Below is a CSi-Locate command file used to locate a 16-bit real-mode application in ROM and RAM (this particular application is not included in any of the SAMP subdirectories, but the actual code used is almost the same as the code in CSILOC\SAMP\MSC16R. Only the file B16RMSC.ASM differs by containing code to initialize 386EX hardware). We will explain what each command in the command file is directing the locator to do.

### .CMD command file

```
debug                  // Create debug file
absolute csamp.abs     // Create OMF86 absolute file
hex    csamp.hex       // Create Intel extended 86 hex file
cpu 386                // Target is a 386, which is
                       // running in real mode as the .EXE
                       // file shows

// These are the classes that need to be raminited
init16r raminit ::  class data class far_data class begdata
+                   class const  class stack class MSG
+                   class BSS

// Locate reset_text Here
locate reset_text :: 84000p

// Locate raminited segments starting at 1000p
locate class far_data :: 1000p

// Locate the remaining segments in ROM at 80000p (code
// segments).
locate init_text * :: 80000p
```

### Explaining the command file

1. The first four commands are explained in the example on pp. 16–17, so they will not be covered here (except to say that if the command file's name would be CSAMP.CMD, then the file arguments to the **ABSOLUTE** and **HEX** commands would be unnecessary).
2. The RAM-initialization command **INIT16R** is used to compress data from the segment classes specified after the "::" into the ROM segment **raminit**. The data is intended to be initialized and located in RAM. The macros mentioned on

*Explaining the command file (continued)*

p. 4 can be used to uncompress and copy the data from the ROM segment to its intended RAM-segment destination.

3. Two of the **LOCATE** commands are used to locate *data* in low RAM (`locate class far_data :: 1000p`), and *code* in higher RAM (`locate init_text * :: 80000p`). The segment map that is part of the .CM conversion map output file, listed on p. 23, shows accurately the respective locations of code and data.

   The other use of **LOCATE** places the reset code at a higher ROM address.

4. The map file generated by the linker on the next page shows the various classes and groups of data segments, as well as the other segments ordered by the linker. If you now look at the located-segment map on p. 23, you will see how locating the **far_data** segment causes all segments in the **DGROUP** to be ordered after it, and how locating **init_text** causes all the remaining segments (except **reset_text**, which was separately located) to be located after it contiguously in memory according to the linker's map-file ordering.

# 16-bit Real-Mode ROMmed Example

***Map file for ROMmed 16-bit real-mode application***

```
 Start  Stop   Length Name                   Class

 00000H 000ADH 000AEH INIT_TEXT             CODE
 000B0H 004B2H 00403H RESET_TEXT            CODE
 004B4H 004F1H 0003EH CMAIN_TEXT            CODE
 004F2H 007DDH 002ECH CUTILS_TEXT           CODE
 007DEH 0348DH 02CB0H _TEXT                 CODE
 03490H 03491H 00002H EMULATOR_TEXT         CODE
 03492H 03492H 00000H C_ETEXT               ENDCODE
 034A0H 034AFH 00010H RAMINIT               INIT
 034B0H 034B0H 00000H EMULATOR_DATA         FAR_DATA
 034B0H 034F1H 00042H NULL                  BEGDATA
 034F2H 039E1H 004F0H _DATA                 DATA
 039E2H 039E2H 00000H XIFCB                 DATA
 039E2H 039E2H 00000H XIFU                  DATA
 039E2H 039E2H 00000H XIFL                  DATA
 039E2H 039E2H 00000H XIFCE                 DATA
 039E2H 039E3H 00002H XIQC                  DATA
 039E4H 039EFH 0000CH DBDATA                DATA
 039F0H 039F0H 00000H XIB                   DATA
 039F0H 039F3H 00004H XI                    DATA
 039F4H 039F4H 00000H XIE                   DATA
 039F4H 03A01H 0000EH CDATA                 DATA
 03A02H 03A02H 00000H XIFB                  DATA
 03A02H 03A02H 00000H XIF                   DATA
 03A02H 03A02H 00000H XIFE                  DATA
 03A02H 03A02H 00000H XPB                   DATA
 03A02H 03A02H 00000H XP                    DATA
 03A02H 03A02H 00000H XPE                   DATA
 03A02H 03A02H 00000H XCB                   DATA
 03A02H 03A02H 00000H XC                    DATA
 03A02H 03A02H 00000H XCE                   DATA
 03A02H 03A02H 00000H XCFB                  DATA
 03A02H 03A02H 00000H XCFCRT                DATA
 03A02H 03A02H 00000H XCF                   DATA
 03A02H 03A02H 00000H XCFE                  DATA
 03A02H 03A02H 00000H XIFM                  DATA
 03A02H 03A1DH 0001CH CONST                 CONST
 03A1EH 03A25H 00008H HDR                   MSG
 03A26H 03B73H 0014EH MSG                   MSG
 03B74H 03B75H 00002H PAD                   MSG
 03B76H 03B76H 00001H EPAD                  MSG
 03B78H 03B8DH 00016H _BSS                  BSS
 03B8EH 03B8EH 00000H XOB                   BSS
 03B8EH 03B8EH 00000H XO                    BSS
 03B8EH 03B8EH 00000H XOE                   BSS
 03B8EH 03B8EH 00000H XOFB                  BSS
 03B8EH 03B8EH 00000H XOF                   BSS
 03B8EH 03B8EH 00000H XOFE                  BSS
 03B90H 03BA1H 00012H c_common              BSS
 03BB0H 083AFH 04800H STACK                 STACK

 Origin   Group
 034B:0   DGROUP
```

```
Real-mode segment map:
   Logical     Physical  Length   Name          Class Group  Memory
-------------------------------------------------------------------

0100:0000     00001000P 00000    EMULATOR_DATA FAR_DATA DGROUP RAM
    :0000     00001000P 00042    NULL          BEGDATA DGROUP RAM
    :0042     00001042P 004f0    _DATA         DATA  DGROUP RAM
    :0532     00001532P 00000    XIFCB         DATA  DGROUP RAM
    :0532     00001532P 00000    XIFU          DATA  DGROUP RAM
    :0532     00001532P 00000    XIFL          DATA  DGROUP RAM
    :0532     00001532P 00000    XIFCE         DATA  DGROUP RAM
    :0532     00001532P 00002    XIQC          DATA  DGROUP RAM
    :0534     00001534P 0000c    DBDATA        DATA  DGROUP RAM
    :0540     00001540P 00000    XIB           DATA  DGROUP RAM
    :0540     00001540P 00004    XI            DATA  DGROUP RAM
    :0544     00001544P 00000    XIE           DATA  DGROUP RAM
    :0544     00001544P 0000e    CDATA         DATA  DGROUP RAM
    :0552     00001552P 00000    XIFB          DATA  DGROUP RAM
    :0552     00001552P 00000    XIF           DATA  DGROUP RAM
    :0552     00001552P 00000    XIFE          DATA  DGROUP RAM
    :0552     00001552P 00000    XPB           DATA  DGROUP RAM
    :0552     00001552P 00000    XP            DATA  DGROUP RAM
    :0552     00001552P 00000    XPE           DATA  DGROUP RAM
    :0552     00001552P 00000    XCB           DATA  DGROUP RAM
    :0552     00001552P 00000    XC            DATA  DGROUP RAM
    :0552     00001552P 00000    XCE           DATA  DGROUP RAM
    :0552     00001552P 00000    XCFB          DATA  DGROUP RAM
    :0552     00001552P 00000    XCFCRT        DATA  DGROUP RAM
    :0552     00001552P 00000    XCF           DATA  DGROUP RAM
    :0552     00001552P 00000    XCFE          DATA  DGROUP RAM
    :0552     00001552P 00000    XIFM          DATA  DGROUP RAM
    :0552     00001552P 0001c    CONST         CONST DGROUP RAM
    :056e     0000156eP 00008    HDR           MSG   DGROUP RAM
    :0576     00001576P 0014e    MSG           MSG   DGROUP RAM
    :06c4     000016c4P 00002    PAD           MSG   DGROUP RAM
    :06c6     000016c6P 00001    EPAD          MSG   DGROUP RAM
    :06c8     000016c8P 00016    _BSS          BSS   DGROUP RAM
    :06de     000016deP 00000    XOB           BSS   DGROUP RAM
    :06de     000016deP 00000    XO            BSS   DGROUP RAM
    :06de     000016deP 00000    XOE           BSS   DGROUP RAM
    :06de     000016deP 00000    XOFB          BSS   DGROUP RAM
    :06de     000016deP 00000    XOF           BSS   DGROUP RAM
    :06de     000016deP 00000    XOFE          BSS   DGROUP RAM
    :06e0     000016e0P 00012    C_COMMON      BSS   DGROUP RAM
    :0700     00001700P 04800    STACK         STACK DGROUP RAM
8000:0000     00080000P 000ae    INIT_TEXT     CODE        ROM
800b:0004     000800b4P 0003e    CMAIN_TEXT    CODE        ROM
800f:0002     000800f2P 002ec    CUTILS_TEXT   CODE        ROM
803d:000e     000803deP 02cb0    _TEXT         CODE        ROM
8309:0000     00083090P 00002    EMULATOR_TEXT CODE        ROM
    :0002     00083092P 00000    C_ETEXT       ENDCODE     ROM
830a:0000     000830a0P 004ad    RAMINIT       INIT        ROM
8400:0000     00084000P 00403    RESET_TEXT    CODE        ROM
```

***Segment-map part of .CM conversion map file for ROMmed 16-bit real-mode application***

Here are the controls to use when preparing your 16-bit real-mode application with Borland tools and CSi-Locate for use with Soft-Scope.

### Borland C/C++ compiler

*Use these controls with the Borland compiler*

- **-v** Debug information.
- **-O-** Disable optimization. You may remove this switch when the module has been debugged.
- **-r-** Don't use register variables.
- **-c** Don't link.

*Example invocation*

```
bcc -v -O- -c  -r- cmain.c
```

### Borland Turbo assembler

*Use these controls with the Borland assembler*

- **/Zi** Provide debug information.
- **/mx or ml** Treat symbols as case sensitive.

*Example invocation*

```
tasm /Zi /mx b16rbcc.asm
```

### Borland TLINK

*Use these controls with the Borland linker*

- **/m** Create a mapfile with publics.
- **/v** Provide debug information.
- **/l** Provide line numbers.
- **/s** Create detailed segment map.
- **/Twe** Create a 16-bit Windows .EXE file.

*Example invocation*

```
tlink /m /s /v /l @csamp.rsp
```

A sample CSAMP.RSP response file is CSILOC\SAMP\BCC16R\CSAMP.FLT.

Here are the controls to use when preparing your 16-bit real-mode application with Microsoft tools and CSi-Locate for use with Soft-Scope. See p. 75 for a note about Microsoft compilers.

***Microsoft C/C++ compiler***

*Use these controls with the Microsoft compiler*

- **/Zi** Include symbolic information. Versions 7 and 8 use **/Z7** to perform this function.
- **/Od** Disable optimization. You may remove this switch when the module has been debugged. It is even possible to leave this switch out, but we recommend you do this only after you are comfortable using Soft-Scope.
- **/Gs** Remove run-time stack probes.
- **/GW** Windows application.
- **/c** Compile only--do not link.

*Example invocation*

```
cl  /Zi /Od /Gs /Gw /c cmain.c
```

***Microsoft LINK***

*Use these controls with the Microsoft linker*

- **/NOD** Ignore default libraries.
- **/NOE** Ignore extended dictionaries.
- **/MAP** Create a map file.
- **/CO** Codeview symbolics.

*Example invocation*

```
link cmain.obj,csamp.exe,csamp.map/MAP/CO/NOD,llibce.lib;
```

***Microsoft ML***

*Use these controls with the Microsoft assembler*

- **/Zd** Include line number information in object file.
- **/Zi** Generate Codeview symbolics in object file.
- **/Cp** Make all symbols case sensitive.
- **/c** Compile only—do not link.

*Example invocation*

```
ml /Zd /Zi /Cp /c b16rmsc.asm
```

Here are the controls to use when preparing your 16-bit real-mode application with Watcom tools and CSi-Locate for use with Soft-Scope.

### Watcom C/C++

*Use these controls with the Watcom compiler*

| | |
|---|---|
| **/s** | Remove stack overflow checking. |
| **/d2** | Create debug information. |

*Example invocation*

```
wcc  /s /d2 cmain.c
```

### Watcom WASM

*Use this control with the Watcom assembler*

| | |
|---|---|
| **-d1** | Create debug information. |

*Example invocation*

```
wasm -d1 b16rwcc.asm
```

### Watcom WLINK

*Use these directives with the Watcom linker*

| | |
|---|---|
| **option map** | Create a map file. |
| **debug all** | Provide full debug information. |
| **format dos** | Create a DOS EXE output file. |
| **name csamp.exe** | Name the output file CSAMP.EXE. |

Rather than attempt to put all the linker directives on one line, you can put them all in an .LNK directive file, one directive per line. The directive file that was used to produce the sample program provided with this software is the file CSILOC\SAMP\WCC16R\CSAMP.FLT.

*Example invocation*

```
wlink @csamp.lnk
```

# 16-bit Protected-Mode Applications

# 3

This chapter covers preparing and locating 16-bit protected-mode applications. There are three examples of such applications supplied with CSi-Locate and they can be found in the following subdirectories:

CSILOC\SAMP\BCC16P (Borland)
CSILOC\SAMP\MSC16P (Microsoft)
CSILOC\SAMP\MSC16PF (Miscrosoft/with Floating point)

This chapter will discuss the example in CSILOC\SAMP\MSC16P to illustrate how you can prepare and locate your own 16-bit protected-mode application.

## Table of Contents

# 16-bit Protected-Mode Example

***Step 1—Compile and link using Microsoft tools. For other tool chains, see the Tools section on pp. 36–38 of this chapter***

We have used batch files to create the sample programs included in all the CSILOC\SAMP subdirectories. For the sake of illustration in this chapter, we will discuss the program found in SAMP\MSC16P, which is composed of one assembly-language file (B16PMSC.ASM), and two C files (CMAIN.C and CUTILS.C), and built with Microsoft tools. The following is the file BUILD.BAT that assembles, compiles, links, and locates the application.

```
:
: Batch file to create csamp.abs
: To use type 'build'
:
: Tools versions used:
:      Microsoft ML v6.10
:      Microsoft C v8.00c (VCPP v1.51)
:      Microsoft Link v5.60.220
:      Concurrent Sciences CSi-Locate 2.02
:

ml /Zd /Zi /Cp /c b16pmsc.asm
cl /Zi /Od /Gs /c /G2 /AS cmain.c
cl /Zi /Od /Gs /c /G2 /AS cutils.c
link /NOD /MAP /CO @csamp.flt;
csiloc csamp.cmd
```

***Step 2—Examine your application's MAP file for class, segment, and group information***

However you build your application, make sure that you direct the linker to generate a map file, because CSi-Locate requires a map file as one of its inputs. We will use the map file on the next page as our reference map file in this illustration.

The detailed segment map allows you to see the classes and segments that you need to locate, and what their relationships are to each other. Since groups are not shown by the Microsoft linker's map file, it is possible for there to be locked-together segments that you are unaware of until after location (see pp. 10–11 for more on this problem).

```
Start     Length     Name                   Class
0001:0000 00010H     raminit                INIT
0002:0000 00010H     sysgdt                 TBL
0003:0000 00010H     sysidt                 TBL
0004:0000 00010H     systss                 TSS
0005:0000 000AEH     init_text              CODE
0005:00B0 00403H     reset_text             CODE
0005:04B4 00224H     _TEXT                  CODE
0006:0000 0003CH     _data                  DATA
0006:003C 00000H     XIFCB                  DATA
0006:003C 00000H     XIFU                   DATA
0006:003C 00000H     XIFL                   DATA
0006:003C 00000H     XIFCE                  DATA
0006:0040 04000H     STACK                  STACK
0006:4040 00000H     CONST                  CONST
0006:4040 00002H     _BSS                   BSS
0006:4050 0000AH     c_common               BSS

Origin   Group
0006:0   DGROUP

Address         Publics by Name              Address         Publics by Value

0005:04B0       hardware_reset               0000:0000  Abs  __acrtused
0005:000E       init_cpp                     0001:0000       raminit_init
0005:00B6       pt_init                      0002:0000       sysgdt_tbl
0001:0000       raminit_init                 0003:0000       sysidt_tbl
0005:0000       start_code                   0004:0000       systss_tss
0005:0004       start_data                   0005:0000       start_code
0005:000A       start_init                   0005:0004       start_data
0005:0006       start_stack                  0005:0006       start_stack
0006:4040       start_tos                    0005:000A       start_init
0002:0000       sysgdt_tbl                   0005:000E       init_cpp
0003:0000       sysidt_tbl                   0005:0027       _boot
0004:0000       systss_tss                   0005:00B6       pt_init
0005:0027       _boot                        0005:04B0       hardware_reset
0005:04EE       _c_data                      0005:04B4       _main
0005:0664       _delay                       0005:04EE       _c_data
0005:05F6       _display_lights              0005:05F6       _display_lights
0006:4052       _lights                      0005:0664       _delay
0005:04B4       _main                        0006:4040       start_tos
0006:4050       _pattern                     0006:4050       _pattern
0000:0000  Abs  __acrtused                   0006:4052       _lights

                                             Program entry point at 0005:0027
```

***Fig. 3.1 Sample MAP file (modified for presentation)***

***Step 3—Create a command file***

Use an ASCII text editor to create a .CMD command file with the commands that direct CSi-Locate to locate your application. Below is a modified version of the file SAMP\MSC16P\CSAMP.CMD. We will begin to explain it in detail in numbered paragraphs immediately following it. The full story of the locator's 38 commands and how to construct command files with them occurs in chapter 5.

***Example .CMD file***

```
//
//  Microsoft C++ 286 protected mode sample command file
//  This file is set up to work with the default
//  configuration of Concurrent Sciences CSiMON
//  debug monitor
//
debug                          // Create debug file
absolute                       // Create OMF286 absolute file
cpu 286                        // Target is 286 running in
                                  // protected mode
locate *::4000p                // Locate first segment at 4000P
gdt sysgdt[3..64]::reserve     // CSiMON needs slots 3 to 64
idt sysidt[0..40]::reserve     // Reserve for Intel CPU
tss286 systss::cs:ip=_boot
+              ss:sp=start_tos
                               // Create an initial TSS
gdt sysgdt::systss             // Place initial tss in GDT
gdt sysgdt::*
integrity *
```

***Explanation of example .CMD file***

1. Note that comments begin with double slashes and end with a carriage return.
2. The placement order of commands is significant. On pp. 54–55 in chapter 5, we provide general guidelines for the relative ordering of the locator's 38 commands in .CMD files. In general, place I/O commands first (the first three commands in the above example are I/O commands).
3. **DEBUG** directs the locator to output symbolic debugging information that is essential for the Soft-Scope debugger. For protected-mode application, the symbolics are placed in the actual absolute output file. For real-mode applications, CSi-Locate puts that information in a .BUG file that it separately creates, with the same name as the .CMD file.

4. **ABSOLUTE** tells the locator to output an absolute file in Intel OMF format. The precise OMF type is determined by the application's mode (real or protected) and the target CPU, which is made known to the locator by means of the **CPU** command. See page 60 for further details on how the locator determines the OMF type.

5. **CPU** specifies the target CPU to the locator. A list of possible values in the 80x86 family is given in table 5.5 on p. 67. This value partly determines the OMF type of absolute files (see 4. above) and is used in other ways by the locator to produce optimal output.

6. **LOCATE** plays perhaps the most crucial role among all the locator commands, for obvious reasons. This command tells the locator explicitly to absolutely locate one or more segments, which in turn may cause other segments to be located. This kind of ripple location effect results from the linker's relative ordering of segments: once the first segment in an ordered collection of segments is located, all subsequent segments as determined by the linking process fall into place. The map file produced by the linker can be used to see how segments are ordered.

   The use of the * (asterisk) here signifies that this command applies to all segments, which actually means that the first segment in the map file's ordering should be located at the address specified. An equivalent command for this application would be the following:

   ```
   locate  raminit ::4000p
   ```

***Step 4—Define protected-mode structures***

7. **GDT** is used to set up the Global Descriptor Table (GDT). The command assumes that a segment to hold the GDT (**sysgdt** in this case) has already been created. For this example, the segment was created by means of the macro **def_tbl** (see p. 4); an alternative way to create the GDT's segment is to use the locator's **CREATE** command (see p. 68).

   The first use of the **GDT** command in the sample .CMD file tells the locator that slots 3–64 are being used for some other purpose, and so are not to be used for this application.

As the comment says, the CSi-Mon monitor requires those slots for its use. The locator will thus begin adding new entries for this application at slot 65.

The second use of the command places the segment **systss**, which is a Task State Segment (TSS) defined with the TSS286 command just above it, into the first available slot, namely, 65. This segment, like **sysgdt**, was also created in the .ASM file with an assembler directive.

The final instance of the command places all remaining segments, that is, all segments other than **systss**, in table slots, starting at 66. You can look in the .CM conversion-map file to see how the remaining segments were placed.

8. **IDT** is used to fill in the Interrupt Descriptor Table (IDT). The command assumes that a segment to hold the IDT (**sysidt** in this case) has already been created. In this example, the segment was created by directive in the .ASM file. The use of the command here tells the locator that some other application is using slots 0–40, so it should begin adding entries, if at all, at slot 41. The comment indicates that the first 41 slots are reserved for use by the CPU.
9. **TSS286** is used to define the segment **systss** as a Task State Segment (TSS), and to assign initial values to the instruction pointer **CS:IP** and the stack pointer **SS:SP**. This command, like **GDT** and **IDT**, assumes that the segment it applies to has already been created. You can see that there is a directive in the file B16PMSC.ASM that creates **systss**.
10. **INTEGRITY** blocks the locator from truncating any segments that have been padded by the linker or compiler.

### *Step 5—Invoke CSi-Locate*

To create a located application, use the following syntax at the DOS prompt:

CSILOC *filename*

where *filename* is the command file (with default extension CMD) that contains the locator commands, as in the following example:

```
csiloc csamp
```

### *.CM conversion map file always output by locator*

If CSi-Locate locates your application as specified without error, the output file(s) that you request with the commands **ABSOLUTE**, **BINARY**, and **HEX** are created, and a conversion-map file *FILENAME*.CM, which is a sort of locator listing file, is also produced. If the locator encounters any errors while trying to locate your application, the only output file is the .CM file, which shows all warnings and error messages generated during the location attempt. Parts of the .CM file produced for our example 16-bit protected-mode application here are given below, with annotations. A more complete .CM is illustrated on pp. 6–7 in chapter 1.

```
Protected-mode segment map:
   Logical     Physical  Length   Name          Class Group
Memory
-------------------------------------------------------------
0210:0000      00004000P 00010    RAMINIT       INIT          ROM
0008:0000      00004010P 00228    SYSGDT        TBL           ROM
0010:0000      00004240P 00148    SYSIDT        TBL           ROM
0208:0000      00004390P 0002c    SYSTSS        TSS           ROM
0218:0000      000043c0P 000ae    INIT_TEXT     CODE          ROM
    :00b0      00004470P 00403    RESET_TEXT    CODE          ROM
    :04b4      00004874P 00224    _TEXT         CODE          ROM
0220:0000      00004aa0P 0003c    _DATA         DATA  DGROUP  ROM
    :003c      00004adcP 00000    XIFCB         DATA  DGROUP  ROM
    :003c      00004adcP 00000    XIFU          DATA  DGROUP  ROM
    :003c      00004adcP 00000    XIFL          DATA  DGROUP  ROM
    :003c      00004adcP 00000    XIFCE         DATA  DGROUP  ROM
    :0040      00004ae0P 04000    STACK         STACK DGROUP  ROM
    :4040      00008ae0P 00000    CONST         CONST DGROUP  ROM
    :4040      00008ae0P 00002    _BSS          BSS   DGROUP  ROM
    :4050      00008af0P 0000a    C_COMMON      BSS   DGROUP  ROM
```

*Should be RAM, not ROM*

[in file CSAMP.CM] Segment map generated by the locator that shows located addresses

# 16-bit Protected-Mode Example

```
Initial GDT:  SYSGDT[0..68]

GDT[0]      0000  Empty           00000000L Lim=00000H DPL=0 gbp av
GDT[1]      0008  Data WR         00004010L Lim=00227H DPL=0 gbP av
            0008:00000000 SYSGDT            TBL

GDT[2]      0010  Data WR         00004240L Lim=00147H DPL=0 gbP av
            0010:00000000 SYSIDT            TBL

GDT[3]      0018  Reserved
GDT[4]      0020  Reserved

...

GDT[63]     01f8  Reserved
GDT[64]     0200  Reserved
GDT[65]     0208  Avail 286 TSS   00004390L Lim=0002bH DPL=0 gbP av
            0208:00000000 SYSTSS            TSS

GDT[66]     0210  Data WR         00004000L Lim=0000fH DPL=0 gbP av
            0210:00000000 RAMINIT           INIT

GDT[67]     0218  Code RD         000043c0L Lim=006d7H DPL=0 gbP av
            0218:00000000 INIT_TEXT         CODE
            0218:000000b0 RESET_TEXT        CODE
            0218:000004b4 _TEXT             CODE

GDT[68]     0220  Data WR         00004aa0L Lim=04059H DPL=0 gbP av
            0220:00000000 _DATA             DATA     DGROUP
            0220:0000003c XIFCB             DATA     DGROUP
            0220:0000003c XIFU              DATA     DGROUP
            0220:0000003c XIFL              DATA     DGROUP
            0220:0000003c XIFCE             DATA     DGROUP
            0220:00000040 STACK             STACK    DGROUP
            0220:00004040 CONST             CONST    DGROUP
            0220:00004040 _BSS              BSS      DGROUP
            0220:00004050 C_COMMON          BSS      DGROUP
```

[In file CSAMP.CM]
Partial map of
initial GDT table

```
Initial IDT:  SYSIDT[0..40]

IDT[0]      00  Reserved
...
IDT[39]     27  Reserved
IDT[40]     28  Reserved
```

[In file CSAMP.CM]
Partial map of
initial IDT table

```
Initial TSS286:  SYSTSS

gdt SYSGDT[65] 0208 Avail 286 TSS  00004390L Lim=0002bH DPL=0 gbP av

AX=0000   BX=0000   CX=0000   DX=0000   SI=0000   DI=0000   BP=0000
DS=0000   ES=0000
LDTR=0000 LINK=0000
CS:IP=0218:0027
SS:SP=0220:4040
SS0:SP0=0000:0000   SS1:SP1=0000:0000   SS2:SP2=0000:0000
FL=0000 [ nt IOPL=0 of df if tf sf zf af pf cf ]
```

[In file CSAMP.CM] Initial TSS data

```
Translating debug symbolics

Debug Symbolics Translation Complete:
    Modules. . . . . 4
    Procedures . . . . 9
    Public Symbols . . 19
    Source Lines . . . 235
    Total Symbols. . . 56
    Types. . . . . 154

Conversion complete, No errors or warnings.
```

[In file CSAMP.CM] Data on symbolics and final line of conversion map file

Here are the controls to use when preparing your 16-bit protected-mode application with Microsoft tools and CSi-Locate for use with Soft-Scope. See p. 75 for a note on Microsoft compilers.

### *Microsoft C/C++ compiler*

*Use these controls with the Microsoft compiler*

- **/Zi** Include symbolic information. Versions 7 and 8 use /Z7 to perform this function.
- **/Od** Disable optimization. You may remove this switch when the module has been debugged. It is even possible to leave this switch out, but we recommend you do this only after you are comfortable using Soft-Scope.
- **/G2** Generate 80286-specific instructions.
- **/Gs** Remove run-time stack probes.
- **/GW** Windows application.
- **/c** Compile only—do not link.

*Example invocation*

```
cl  /Zi /Od /G2 /Gs /Gw /c cmain.c
```

### *Microsoft LINK*

*Use these controls with the Microsoft linker*

- **/NOD** Ignore default libraries.
- **/MAP** Create a map file.
- **/CO** Codeview symbolics.

*Example invocation*

```
link  /MAP/CO/NOD @csamp.flt
```

Here is an example .FLT file (\SAMP\MSC16P\CSAMP.FLT):

```
b16pmac.obj+
cmain.obj+
cutils.obj
csamp.exe,csamp.map,,csamp.def;
```

The file \SAMP\B16PMSC\CSAMP.DEF contains the following further linker specifications:

```
NAME CSAMP
DESCRIPTION 'CSAMP'
EXETYPE WINDOWS
CODE PRELOAD FIXED
DATA PRELOAD FIXED
STACKSIZE 16384       ; Make this match the
                      ; stack size in startup
                      ; code, which is 16KB
```

**Microsoft ML**

*Use these controls with the Microsoft assembler*

| | |
|---|---|
| **/Zd** | Include line number information in object file. |
| **/Zi** | Generate Codeview symbolics in object file. |
| **/Cp** | Make all symbols case sensitive. |
| **/c** | Compile only—do not link. |

*Example invocation*

```
ml /Zd /Zi /Cp /c b16pmsc.asm
```

Here are the controls to use when preparing your 16-bit protected-mode application with Borland tools and CSi-Locate for use with Soft-Scope.

### Borland C/C++ compiler

*Use these controls with the Borland compiler*

- **-v** Debug information.
- **-2** Generate 16-bit 80286 protected-mode instructions.
- **-O-** Disable optimization. You may remove this switch when the module has been debugged.
- **-r-** Don't use register variables.
- **-c** Don't link.

*Example invocation*

```
bcc -v -2 -O- -c  -r- cmain.c
```

### Borland Turbo assembler

*Use these controls with the Borland assembler*

- **/Zi** Provide debug information.
- **/mx or ml** Treat symbols as case sensitive.

*Example invocation*

```
tasm /Zi /mx b16pbcc.asm
```

### Borland TLINK

*Use these controls with the Borland linker*

- **/m** Create a mapfile with publics.
- **/v** Provide debug information.
- **/l** Provide line numbers.
- **/s** Create detailed segment map.
- **/Twe** Create a 16-bit Windows .EXE file

*Example invocation*

```
tlink /m /s /v /l @csamp.rsp
```

A sample CSAMP.RSP response file is CSILOC\SAMP\BCC16P\CSAMP.FLT.

# 32-bit Protected-Mode Applications

4

This chapter covers preparing and locating 32-bit protected-mode applications. There are two examples of such applications supplied with CSi-Locate and they can be found in the following subdirectories:

CSILOC\SAMP\WCC32P (Watcom)
CSILOC\SAMP\MSC32P (Microsoft)

This chapter will discuss the example program in CSILOC\SAMP\MSC32P to illustrate how you can prepare and locate your own 32-bit protected-mode application.

## Table of Contents

***Step 1—Compile and link using Microsoft tools. For other tool chains, see the Tools section on pp. 49–50 of this chapter***

We have used batch files to create the sample programs included in all the CSILOC\SAMP subdirectories. For the sake of illustration in this chapter, we will discuss the program found in SAMP\MSC32P, which is composed of one assembly-language file (B32FPMSC.ASM) and two C files (CMAIN.C and CUTILS.C), and built with Microsoft tools. The following is the file BUILD.BAT that assembles, compiles, links, and locates the application.

```
:
: Batch file to create csamp.abs
: To use type 'build'
:
: Tools versions used:
:      Microsoft ML v6.10
:      Microsoft 32-Bit Compiler v8.00
:      Microsoft 32-Bit Executable Linker v1.00
:      Concurrent Sciences CSi-Locate 2.02
:

ml  /Fl /c b32fpmsc.asm
cl /c /Zi /Od /G3 /X cmain.c
cl /c /Zi /Od /G3 /X cutils.c
link @csamp.flt
csiloc csamp.cmd
```

Our primary focus in this chapter will be on the complex command file CSAMP.CMD ON p. 42, but before we scrutinize it, we will discuss the map file that plays an important role in the location process.

***Step 2—Examine your application's MAP file for class, segment, and group information***

When you build your application, you need to direct your linker to generate a map file because CSi-Locate requires one as one of its inputs. The map file on the next page was generated as part of the build process defined above. The detailed segment map allows you to see the classes and segments (see pp. 10–11 for a word about the lack of group information here) that you need to locate, and what their relationships are to each other.

***Paging used with this flat-model application***

The Microsoft 32-bit compiler can build only flat-model applications. If you want to separate segments into RAM and ROM, you will need to use paging, which is covered in this example. Paging requires roughly 4-percent execution-time overhead.

*Fig. 4.1 Complete Sample MAP file [CSAMP.MAP]*

```
csamp

 Timestamp is 30b1024e (Mon Nov 20 14:33:50 1995)

 Preferred load address is 00010000

 Start         Length     Name                   Class
 0001:00000000 000000e1H INIT_TEX               CODE
 0002:00000000 00000265H .text                  CODE
 0003:00000000 00000014H .bss                   DATA
 0004:00000000 00000063H .rdata                 DATA
 0005:00000000 00000048H .data                  DATA
 0006:00000000 00004000H STACK                  DATA
 0007:00000000 0000003aH .edata                 DATA
 0009:00000000 00000a02H .debug$C               DATA
 0009:00000a04 00000000H .debug$G               DATA
 0009:00000a04 00000008H .debug$H               DATA
 0009:00000a0c 0000035aH .debug$S               DATA
 0009:00000d66 00000048H .debug$T               DATA

  Address         Publics by Value              Rva+Base   Lib:Object

 0001:00000000       start_code                 00011000   b32fpmsc.obj
 0001:00000006       start_data                 00011006   b32fpmsc.obj
 0001:00000008       start_stack                00011008   b32fpmsc.obj
 0001:0000000e       start_init                 0001100e   b32fpmsc.obj
 0001:00000014       pbbr                       00011014   b32fpmsc.obj
 0001:00000018       temp                       00011018   b32fpmsc.obj
 0001:0000001c       _boot                      0001101c   b32fpmsc.obj
 0001:0000001c       _start                     0001101c   b32fpmsc.obj
 0002:00000000       _main                      00012000 f cmain.obj
 0002:00000048       _c_data                    00012048 f cutils.obj
 0002:00000176       _display_lights            00012176 f cutils.obj
 0002:000001d5       _delay                     000121d5 f cutils.obj
 0003:00000000       _lights                    00013000   <common>
 0003:00000008       _pattern                   00013008   <common>
 0007:00000000       CSAMP_MSC_EXPORTS          0001a000   csamp.exp
 0006:00004000       start_tos                  0001a000   b32fpmsc.obj

 entry point at        0001:0000001c

 Static symbols

 0002:00000201       _delay_fine                00012201 f cutils.obj
 0002:0000022e       _strcpy                    0001222e f cutils.obj

FIXUPS: 2125 23 a8 32 fffffe02 e 8
```

*Fig. 4.2. Command file CSAMP.CMD*

```
//
//  Microsoft C++ 386 32bit flat model sample command file
//  This file is set up to work with the default
//  configuration of Concurrent Sciences CSi-MON
//  debug monitor
//

exec csamp.exp
debug                                   // Create debug file
absolute                                // Create OMF386 absolute file
cpu 386                                 // Target is 386 running in protected mode
create dir_name                         // Create segment for page directory
create page_table                       // Create segment for page table
create raminit                          // Create segment for RAM init table
init32p raminit :: class data           // Build init table for data segments

                                        // set directory[0] to point to page_table
pagedirectory dir_name[0]::page_table
locate .text :: 8000p                   // Locate first segment at 8000 physical
locate .text :: 80001                   // Locate physical/linear at same location
locate .data ::11000p                   // Locate data segment at 11000 physical
create systss                           // Create segment for initial TSS
create sysgdt                           // Create segment for GDT
create sysidt                           // Create segment for IDT
gdt sysgdt[3..64] :: reserve            // CSi-MON needs slots 3 to 64
idt sysidt[0..40] :: reserve            // Reserve for Intel CPU
tss386 systss :: cs:eip=_boot           // Create an inital TSS
+                ss:esp=start_tos
fixup selector start_data = group flat_d
fixup far32 start_code = _main
fixup far32 start_stack = start_tos
fixup far32 start_init = raminit
fixup physical pbbr = dir_name
create csimon_rom::limit=5000h          // Create alias segment over csi-mon rom area
ram csimon_rom                          // Don't generate data in HEX file—CSi-Mon
                                              // is already on target
locate csimon_rom::70000p               // <- Check where CSi-Mon rom area is located
locate csimon_rom::700001

create csimon_ram::limit=2000h          // Create alias segment over csi-mon ram area
ram csimon_ram                          // Don't create segment in HEX file—CSi-Mon
                                              // is already on target
locate csimon_ram::0p                   // <- Check where CSi-Mon ram area is located
locate csimon_ram::01                   // Start at 0 because monitor writes into
                                              // vector table

gdt sysgdt :: *                         // Put all segment descriptors in GDT
pagetable dir_name::*                   // Build page table entries for all
                                                    // segments
```

***Step 3—Create a command file***

Use an ASCII text editor to create a .CMD command file with the commands that direct CSi-Locate to locate your application. On the opposite page is the command file SAMP\MSC32P\CSAMP.CMD. We will begin to explain it in detail in numbered paragraphs just below. The full story of the locator's 38 commands and how to construct command files with them occurs in chapter 5.

1. Note that comments begin with double slashes and end with a carriage return.

2. The order in which commands occur in the command file is significant. You can think of CSi-Locate as though it were an interpreter that processed each command as it read it in. To help you get started with ordering commands, we provide general guidelines for command ordering in .CMD files on pp. 54–55. In general, place I/O commands first (the first four commands in the above example are I/O commands) and location and table-construction commands last.

3. The **EXEC** command identifies an alternate to the default input executable. The default's name matches the .CMD file's name and its extension is EXE.

4. **DEBUG** directs the locator to output symbolic debugging information that is essential for the Soft-Scope debugger. For protected-mode applications, the symbolics are placed in the actual absolute output file. For real-mode applications, CSi-Locate puts that information in a .BUG file that it separately creates, with the same name as the .CMD file.

5. **ABSOLUTE** tells the locator to output an absolute file in Intel OMF format. The precise OMF type is determined by the application's mode (real or protected) and the target CPU, which is made known to the locator by means of the **CPU** command. See page 60 for further details on how the locator determines the OMF type.

6. **CPU** specifies the target CPU to the locator. A list of possible values in the 80x86 family is given in table 5.5 on p. 67. This value partly determines the OMF type of absolute files (see 5. above) and is used in other ways by the locator to produce optimal output.

7. **CREATE** creates a segment. It is useful for creating segments for paging apparatus (page directory and page tables), protected-mode structures (TSSs, GDT, IDT, and LDT), RAM-initialization code (RAMINIT), and for alias segments that are used to accommodate preexisting segments (for example, the segments of the CSi-Mon monitor in both RAM and ROM).

8. The **INIT32P** *seg_name* :: *seg_list* command specifies that the segment *seg_name* will contain compressed data from the segments in *seg_list*. This is useful for read-write data that you want to be initialized at boot-up time. If you don't have some place to store initialized data in ROM and then copy it into RAM, all RAM-based data in your program will be zeroed out. There are macros provided with CSi-Locate, in file CSILOC.INC, that compress and then unpack and copy data from ROM to RAM. For more details on these, see p. 4.

9. **PAGEDIRECTORY** identifies page-directory entries. **`dir_name`** in the example is the page-directory segment, and the first page table is here specified to be the segment **`page_table`**.

10. **LOCATE** plays perhaps the most crucial role among all the locator commands, for obvious reasons. This command tells the locator explicitly to absolutely locate one or more segments, which in turn may cause other segments to be located. This kind of ripple location effect results from the linker's relative ordering of segments: once the first segment in an ordered collection of segments is located, all subsequent segments as determined by the linking process fall into place. The map file produced by the linker can be used to see how segments are ordered.

    The first use locates the **`.TEXT`** segment at 8000P and then the next use locates the same segment at 8000L. This double locating of the same segment is used to bring physical and linear addresses into alignment. Since **`INIT_TEXT`** is in the class **`CODE`** with **`.TEXT`**, it too is located with this command, but before **`.TEXT`** as it occurs in the ordering determined by the linker, as shown in the map file.

The **DATA** segment is located apart from the code.

11. **GDT** is used to fill in the Global Descriptor Table (GDT). The command assumes that a segment to hold the GDT (**sysgdt** in this case) has already been created. For this example, the segment was created by a use of the locator's **CREATE** command, though it is equally possible to explicitly create the segment with a directive in an assembly file (see p. 4).

    The first use of the command in the sample .CMD file tells the locator that slots 3–64 are being used for some other purpose, and so are not to be used for this application. As the comment says, the CSi-Mon monitor requires those slots for its use. The locator will thus begin adding new entries for this application at slot 65.

    The second use of the command places into the table beginning at slot 65 all segments not already placed there, which in this case is all segments since none has yet been placed there. You can look in the .cm conversion-map file to see how the segments were placed. That part of the CM file occurs just below on p. 47.

12. **IDT** is used to fill in the Interrupt Descriptor Table (IDT). The command assumes that a segment to hold the IDT (**sysidt** in this case) has already been created. In this example, the segment was created by the locator as a result of a **CREATE** command. The use of the command here tells the locator that some other application is using slots 0–40, so it should begin adding entries, if at all, at slot 41. The comment indicates that the first 41 slots are reserved for use by the CPU.

13. **TSS386** is used to define the segment **systss** as a Task State Segment (TSS), and to assign initial values to the instruction pointer **CS:EIP** and the stack pointer **SS:ESP**. This command, like **GDT** and **IDT** assumes that the segment it applies to has already been created; **CREATE** was used in this case to make that segment.

***Step 4—Define protected-mode structures***

14. **FIXUP** provides a way for you to modify segments. The examples here add a segment to a group, install addresses at various memory locations, and set up the page table directory.
15. **RAM** is used to exclude segments from being output. Since alias segments are really just place holders for preexisting segments and are not truly on a par with the other types of segments created for code and data, they are best excluded from inclusion with the application's bona fide segments. That is what happens here with the two uses of the **RAM** command.
16. Lastly, the **PAGETABLE** command builds page-table entries for all segments, including alias segments.

***Step 5—Invoke CSi-Locate***

To create a located application, use the following syntax at the DOS prompt:

CSILOC *filename*

where *filename* is the command file (with default extension CMD) that contains the locator commands, as in the following example:

```
csiloc csamp
```

***A .CM conversion map file always output by locator***

If CSi-Locate locates your application as specified without error, the output file(s) that you request with the commands **ABSOLUTE**, **BINARY**, and **HEX** are created, and a conversion-map file *FILENAME*.CM, which is a sort of locator listing file, is also produced. If the locator encounters any errors while trying to locate your application, the only output file is the .CM file, which shows all warnings and error messages generated during the location attempt. Parts of the .CM file produced for our example 32-bit protected-mode application here are given below, with annotations. A more complete .CM is illustrated on pp. 6–7 in chapter 1.

```
Protected-mode segment map:
   Logical    Linear    Physical  Length   Name          Class Group  Memory
-------------------------------------------------------------------------------
0240:0000     00000000L 00000000P 02001    CSIMON_RAM                 RAM
0208:00000000 00007000L 00007000P 000000e1 INIT_TEX      CODE  FLAT_C ROM
    :00001000 00008000L 00008000P 00000265 .TEXT         CODE  FLAT_C ROM
0210:00002000 00009000L 00009000P 00000014 .BSS          DATA  FLAT_D RAM
    :00003000 0000a000L 0000a000P 00000063 .RDATA        DATA  FLAT_D RAM
    :00004000 0000b000L 00011000P 00000048 .DATA         DATA  FLAT_D RAM
    :00005000 0000c000L 0000c000P 00004000 STACK         DATA  FLAT_D RAM
    :00009000 00010000L 00010000P 0000003a .EDATA        DATA  FLAT_D RAM
0218:0000     00013000L 00013000P 01000    DIR_NAME                   ROM
0220:0000     00014000L 00014000P 01000    PAGE_TABLE                 ROM
0228:0000     00015000L 00015000P 00102    RAMINIT                    ROM
0230:0000     00015110L 00015110P 00068    SYSTSS                     ROM
0008:0000     00015180L 00015180P 00248    SYSGDT                     ROM
0010:0000     000153d0L 000153d0P 00148    SYSIDT                     ROM
0238:0000     00070000L 00070000P 05001    CSIMON_ROM                 RAM
```

***Figure 4.3. Conversion map file***

[In CSAMP.CM] Segment map that shows located segment addresses

```
Initial GDT:  SYSGDT[0..72]
-------------------------------------------------------------------------------
GDT[0]      0000  Empty          00000000L Lim=00000H DPL=0 gbp av
GDT[1]      0008  Data WR        00015180L Lim=00247H DPL=0 gbP av
            0008:00000000 SYSGDT

GDT[2]      0010  Data WR        000153d0L Lim=00147H DPL=0 gbP av
            0010:00000000 SYSIDT

GDT[3]      0018  Reserved
...
GDT[64]     0200  Reserved
GDT[65]     0208  Code RD        00007000L Lim=0bfffH DPL=0 gBP av
            0208:00000000 INIT_TEX          CODE     FLAT_C
            0208:00001000 .TEXT             CODE     FLAT_C

GDT[66]     0210  Data WR        00007000L Lim=0bfffH DPL=0 gBP av
            0210:00002000 .BSS              DATA     FLAT_D
            0210:00003000 .RDATA            DATA     FLAT_D
            0210:00004000 .DATA             DATA     FLAT_D
            0210:00005000 STACK             DATA     FLAT_D
            0210:00009000 .EDATA            DATA     FLAT_D

GDT[67]     0218  Data WR        00013000L Lim=00fffH DPL=0 gbP av

GDT[68]     0220  Data WR        00014000L Lim=00fffH DPL=0 gbP av

GDT[69]     0228  Data WR        00015000L Lim=00101H DPL=0 gbP av

GDT[70]     0230  Avail 386 TSS  00015110L Lim=00067H DPL=0 gbP av
            0230:00000000 SYSTSS

GDT[71]     0238  Empty          00070000L Lim=05000H DPL=0 gbP av
            0238:00000000 CSIMON_ROM

GDT[72]     0240  Empty          00000000L Lim=02000H DPL=0 gbP av
            0240:00000000 CSIMON_RAM
```

[In CSAMP.CM] Partial initial GDT

# 32-bit Protected-Mode Example

```
Initial IDT:  SYSIDT[0..40]

IDT[0]      00  Reserved
...
IDT[40]     28  Reserved
```

[In CSAMP.CM]
Initial IDT

```
Initial TSS386:  SYSTSS

gdt SYSGDT[70] 0230 Avail 386 TSS  00015110L Lim=00067H DPL=0 gbP av

   EAX=00000000    EBX=00000000    ECX=00000000    EDX=00000000
   ESI=00000000    EDI=00000000    EBP=00000000
   DS=0000         ES=0000         FS=0000         GS=0000
   LDTR=0000       LINK=0000
   CS:EIP=0208:0000001C
   SS:ESP=0210:00009000
   SS0:ESP0=0000:00000000
   SS1:ESP1=0000:00000000
   SS2:ESP2=0000:00000000
   EFL=00000000 [ ac vm rf nt IOPL=0 of df if tf sf zf af pf cf ]
   CR3=00000000 [ PDBR=0 pcd pwt ]
   IO_MAP=0000
   TRAP=0
```

[In CSAMP.CM]
Initial TSS

```
Page Directory: DIR_NAME at 00013000P

[0]             00000000L 00014000P [ d a us RW P ]  PAGE_TABLE
[1..1023]       00400000L Empty
```

[In CSAMP.CM]
Page Directory

```
Page Table from DIR_NAME[0]:  PAGE_TABLE at 00014000P

[0][0..1]       00000000L 00000000P [ d a us RW P ]  CSIMON_RAM
[0][2..6]       00002000L Empty
[0][7]          00007000L 00007000P [ d a us rw P ]  INIT_TEX CODE
[0][8]          00008000L 00008000P [ d a us rw P ]  .TEXT CODE
[0][9]          00009000L 00009000P [ d a us RW P ]  .BSS DATA
[0][10]         0000a000L 0000a000P [ d a us rw P ]  .RDATA DATA
[0][11]         0000b000L 00011000P [ d a us RW P ]  .DATA DATA
[0][12..15]     0000c000L 0000c000P [ d a us RW P ]  STACK DATA
[0][16]         00010000L 00010000P [ d a us rw P ]  .EDATA DATA
[0][17..18]     00011000L Empty
[0][19]         00013000L 00013000P [ d a us RW P ]  DIR_NAME
[0][20]         00014000L 00014000P [ d a us RW P ]  PAGE_TABLE
[0][21]         00015000L 00015000P [ d a us RW P ]  RAMINIT
[0][22..111]    00016000L Empty
[0][112..116]   00070000L 00070000P [ d a us RW P ]  CSIMON_ROM
[0][117..1023]  00075000L Empty
```

[In CSAMP.CM]
Page Table

Here are the controls to use when preparing your 32-bit protected-mode application with Microsoft tools and CSi-Locate for use with Soft-Scope. See p. 75 for a note on Microsoft compilers.

*Use these controls with the Microsoft compiler*

***Microsoft C/C++ compiler***

- **/Zi** Include symbolic information. Versions 7 and 8 use **/Z7** to perform this function.
- **/Od** Disable optimization. You may remove this switch when the module has been debugged. It is even possible to leave this switch out, but we recommend you do this only after you are comfortable using Soft-Scope.
- /c Compile only—do not link.

*Example invocation*

```
cl  /Zi /Od /Gs /Gw /c cmain.c
```

*Use these controls with the Microsoft 32-bit Executable Linker*

***Microsoft LINK***

- **/NODEFAULTLIB** Ignore default libraries.
- **/MAP** Create a map file.
- **/DEBUG** Provide symbolics.

*Example invocation*

```
link  @csamp.flt
```

See CSILOC\SAMP\MSC32P\CSAMP.FLT for an example of this directive file.

*Use these controls with the Microsoft assembler*

***Microsoft ML***

- **/Zd** Include line number information in object file.
- **/Zi** Generate Codeview symbolics in object file.
- **/Cp** Make all symbols case sensitive.
- /c Compile only—do not link.

*Example invocation*

```
ml /Zd /Zi /Cp /c b32fpmsc.asm
```

Here are the controls to use when preparing your 32-bit protected-mode application with Watcom tools and CSi-Locate for use with Soft-Scope.

### Watcom C/C++

*Use these controls with the Watcom compiler*

| | |
|---|---|
| **/s** | Remove stack overflow checking. |
| **/d2** | Create debug information. |

*Example invocation*

```
wcc  /s /d2 cmain.c
```

### Watcom WASM

*Use this control with the Watcom assembler*

| | |
|---|---|
| **-d1** | Create debug information. |

*Example invocation*

```
wasm -d1 b32pwcc.asm
```

### Watcom WLINK

*Use these directives with the Watcom linker*

| | |
|---|---|
| **option map** | Create a map file. |
| **option dosseg** | Order segments in special way. |
| **debug all** | Provide full debug information. |
| **format phar seg** | Create a segmented Phar Lap EXP output file. |
| **name csamp.exp** | Name the output file CSAMP.EXP. |

Rather than attempt to put all the linker directives on one line, you can put them all in an .LNK directive file, one directive per line. The directive file that was used to produce the sample program provided with this software is the file CSILOC\SAMP\WCC32P\CSAMP.FLT.

*Example invocation*

```
wlink @csamp.lnk
```

# Command Reference 5

This chapter describes CSi-Locate's 38 commands and how you can use them to locate your application in precisely the way that you want it located.

The chapter begins with a description of the .CMD command file that you build to contain commands that CSi-Locate follows to locate your application. Then there is a set of command-ordering guidelines that are intended to assist you in organizing the commands in your .CM command file. They are more heuristic in nature than they are hard-and-fast rules.

Then you will find a two-page summary of the syntax element in commands, followed by a two-page summary listing of the locator's 38 commands. The remainder of the chapter consists of an alphabetically ordered command-reference section containing a detailed explanation of each command.

## Table of Contents

CSi-Locate uses a sequential command file to control processing action. Here are some of its general characteristics:

- The default command-file extension is CMD.
- The **VERBOSE** command, which is used to provide extra information to you about what CSi-Locate is doing, can occur anywhere in the command file. Turn on verbose mode by adding the keyword **ON** to the command, and off by adding **OFF**. Below is a part of a .CM file that exemplifies the sort of messages that you receive in verbose mode:

```
[20] verbose on
[21] create startjmp :: limit = 5
    >>>  Created segment: STARTJMP.
    >>>  LIMIT=00000005 for STARTJMP
[22] fixup byte startjmp:0 = 0xea
[23] fixup far16 startjmp:1 = main
[24] locate startjmp :: 0FFFF0p
    >>>  STARTJMP located at 000FFFF0L.
```

- Commands that locate classes locate the entire class contiguously according to the linker's default ordering of segments within the class. If you want to locate a segment separately from the rest of its class, you must place locating commands for that segment before commands that locate the rest of the class or use the EXCEPT keyword.
- With the exception of public-symbol names, whose characters must exactly match in case the names used to declare them in application files, all names, identifiers, prefixes, and suffixes in command files are not case sensitive. Thus, the following examples are equivalent:

```
CREATE MY_DATA :: LIMIT=0X50
create my_data :: limit=0x50
```

This may cause problems if you have symbols that differ only in case and you compile and link with a case-sensitivity switch on.

- Blank lines and other white space are ignored and can be used however you want.

- Maximum command-line length is 222 characters.

- Commands may span multiple lines. To continue a command on subsequent lines, use a plus sign (+) as the first character on each continuation line:

```
tss386 tss_xxx :: cs:eip=main, ds=data_seg,
+                 fl.if=0x1
```

- Comments can be placed anywhere in the command file. Use double slashes to start a comment; a comment ends at the end of the line that it starts on:

```
// This is a sample comment line
cpu 386  //This is another sample comment,
         //which spans two lines
```

- Command files should be structured according to the command-grouping guidelines given on the following pages (54–55). These guidelines are not hard and fast rules for command-file construction, as some of the example command files in earlier chapters testify to. In learning to use CSi-Locate's 38 commands, you can use the guidelines to provide order to an otherwise seemingly random command-file construction process.

***Recommended ordering of commands in command file***

We recommend that you order the commands in the .CMD command file according to the following groupings. Place commands in Group-1 first, then place commands in Group-*N+1* after commands in Group-*N*. Commands within the same group can be ordered in any way you want.

Though these are just recommended guidelines, we strongly urge that you learn to build your own locator command files by following them. Don't be surprised if you notice that the example command files discussed in chapters 2–4 don't follow these guidelines strictly. They abide by the essential rules, but may diverge for less important ones.

Only commands in Group-1 absolutely must be placed before commands in Group-3 and above, and Group-6 commands should occur last. The Group-0 command can occur freely in the file (see p. 94 below for more on the **VERBOSE** command).

***VERBOSE can occur freely throughout the .CMD file***

Group-0 [CSi-Locate Debug Information]

```
VERBOSE
```

Group-1 [Input]

```
CPU
EXEC
MAP
```

Group-2 [Output]

```
ABS
BIN
DEBUG
HEX
PRINT
```

Group-3 [Segment Creation, Definition, and Alteration]

```
ALIAS
ATTRIBUTE
CREATE
```

Group-3 [Segment Creation, Definition, and Alteration] (*Continued*)

FIXUP
INIT16P
INIT16R
INIT32P
PAGEDIRECTORY
PMODE
RAM
RMODE

Group-4 [Segment Location]

BASE
INTEGRITY
LOCATE
ROMBASE
ROMMOVE

Group-5 [Protected-Mode Structures]

CALL286
CALL386
INT286
INT386
TASKGATE
TRAP286
TRAP386
TSS286
TSS386

Group-6 [Table Constructors]

GDT
IDT
LDT
PAGETABLE

***Follow these grouping guidelines when you build your .CMD command file***

**Table 5.1. Command syntax elements**

| | |
|---|---|
| * | Signifies all other segments that have not already been explicitly located, modified, etc. |
| \| | Separates mutually exclusive alternatives. |
| ( ) | Enclose alternative entries (separated by "\|"), as in the following example:<br>CS=(*number* \| *seg_name* \| *pub_sym*)<br>is equivalent to,<br>CS=*number* \| CS=*seg_name* \| CS=*pub_sym* |
| [ ] | Enclose optional entries. |
| *addressL* | *hex_num*L {linear address} |
| *addressP* | *hex_num*P {physical address} |
| *assign* | *field=(ptr_value \| seg_value \| num_value)* |
| *assign_list* | See tables 5.7 and 5.8 on pp. 92–93. |
| *attribute_list* | See table 5.3 on p. 62. |
| *class_name* | Character string that identifies a class. |
| *cpu_name* | See table 5.5 on p. 67. |
| *dec_num* | Decimal number. |
| *dir_name* | Name of segment where page-table directory is located. |
| *filename* | DOS filename with optional extension. |
| *gate_options* | DPL=*number* \| COUNT=*number* \| (PRESENT \| NOTPRESENT) \| ENTRY=*ptr_value* |
| *group_name* | Character string that identifies a group. |
| *hex_num* | Hex number. Must have prefix 0x (or 0X) or suffix H (or h). Numbers that begin with a letter (a..f) must have a zero (0) prefix. If more than eight numbers are given, the eight least significant digits are used. |
| *kind* | See table 5.6 on p. 71. |
| *num_value* | (OFFSETOF *pub_sym)* \| *number* |
| *number* | *hex_num* \| *dec_num* |
| *ptr_value* | *pub_sym* \| *seg_value* \| *num_value* |
| *pub_sym* | Public symbol listed in the map file. |
| *range* | [*number1*[..*number2*]]<br>Beginning and ending brackets are required. If *number2* is omitted, range has length 1 starting at *number1*. |

| | |
|---|---|
| *seg_list* | Segment list containing segments, classes, and groups, arranged in any order and used as many times as you want. Specify elements in any of the following ways (optional commas can be used to separate entries, as in first line below) :<br>*seg_name_opt* [[,] *seg_name_opt*]*<br>GROUP *group_name* [EXCEPT *seg_name_opt* [*seg_name_opt*]*]<br>CLASS *class_name* [EXCEPT *seg_name_opt* [*seg_name_opt*]*]<br>* [EXCEPT *seg_name_opt* [*seg_name_opt*]*] |
| *seg_name* | Character string that identifies a segment. |
| *seg_name_opt* | [SEGMENT] *seg_name* |
| *seg_value* | SEGMENTOF *pub_sym*<br>GROUP *group_name*<br>*seg_name_opt* |
| *selector:offset* | A logical address consisting of two hex numbers separated by a colon. Hex-number suffix or prefix is not required, that is, any number before or after a colon is automatically interpreted as a hex number. |

***Keywords used in locator commands***

| | |
|---|---|
| CLASS | Indicates following name is a class name |
| COUNT | Gate-descriptor word count |
| DPL | Gate-descriptor privilege level |
| ENTRY | Gate entry point |
| EXCEPT | Indicates exclusion of following segment(s), class(es), or group(s). |
| GROUP | Indicates following name is a group name |
| LENGTH | Indicates following number is the number of bytes after the public symbol that PMODE or RMODE applies to |
| NOTPRESENT | Signifies gate-descriptor present flag is false |
| OFF | Signifies end of verbose mode |
| OFFSETOF | Indicates offset of following public symbol |
| ON | Signifies start of verbose mode |
| PRESENT | Signifies gate-descriptor present flag is true |
| RESERVE | Indicates descriptor-table entries are reserved |
| SEGMENT | Indicates following name is a segment name |
| SEGMENTOF | Indicates segment of following public symbol |

***Table 5.2. Command syntax***

ABS[OLUTE] [*filename*]

ALIAS *seg_name_opt1* :: (*seg_name_opt2* | (GROUP *group_name* ))

ATTRIBUTE *seg_list* :: *attribute_list*

BASE *seg_list* :: *addressL*

BIN[ARY] [*filename*]

CALL286 *seg_list* :: *gate_options*
CALL386 *seg_list* :: *gate_options*

CPU *cpu_name*

CREATE *seg_name* [:: *attribute_list*]

DEBUG [*filename*]

EXEC *filename*

FIXUP *kind ptr_value* =(*ptr_value* | *num_value* | *seg_value* )

GDT *seg_name* [*range*] [:: (RESERVE | *seg_list* )]

HEX [*filename*]

IDT *seg_name* [*range*] [:: (RESERVE | *seg_list* )]

INIT16P *seg_name* :: *seg_list*
INIT16R *seg_name* :: *seg_list*
INIT32P *seg_name* :: *seg_list*

INT286 *seg_list* :: *gate_options*
INT386 *seg_list* :: *gate_options*

INTEGRITY*seg_list*

LDT *seg_name* [*range* ] [:: (RESERVE | *seg_list* )]

LOCATE *seg_list* :: (*addressL* | *addressP*)

MAP *filename*

PAGEDIRECTORY *dir_name* [*range*] :: *seg_list*

PAGETABLE *dir_name* :: *seg_list*

PMODE *seg_list* |
(*pub_sym1* ((to *pub_sym2* ) | (LENGTH *number* )))

PRINT

RAM *seg_list*

RMODE *seg_list* |
(*pub_sym1* ((to *pub_sym2* ) | (LENGTH *number* )))

ROMBASE *seg_list* :: *addressP*

ROMMOVE *seg_list* :: *addressP*

TASKGATE *seg_list* :: *gate_options*

TRAP286 *seg_list* :: *gate_options*
TRAP386 *seg_list* :: *gate_options*

TSS286 *seg_list* :: *assign_list*
TSS386 *seg_list* :: *assign_list*

VERBOSE (ON | OFF)

***Italicized syntax elements***

For an explanation of the italicized syntax elements (e.g., *seg_list)*, see pp. 56–57.

*Output command*

## ABS[OLUTE] [*filename*]

This command creates an .ABS output file and can also be used to change the file's name, which by default is the same as the .CMD command file input to CSi-Locate.

- By default, no absolute, binary, or hex file is output. You must use one of the three output commands (**ABSOLUTE**, **BINARY**, or **HEX**) to generate output.
- These output commands can be used one at a time, all together, or in any combination. Each command will generate one output file.
- The output format of absolute files is a function of both the **CPU** command and the presence of protected-mode structures, according to the following conditions:.

  OMF386: CPU >= 386 & protected-mode structures present
  OMF286: CPU = 286 & protected-mode structures present
  OMF86: No protected-mode structures

  These various OMF formats are extensions of Intel OMF created by Concurrent Sciences for use with the Soft-Scope source level debugger (see p. 8).

*Examples*

```
absolute csamp.abs
absolute
absolute my_file.out
```

## ALIAS [SEGMENT] *seg_name1* :: (([SEGMENT] *seg_name2*) | (GROUP *group_name*))

*Segment-modification command*

This command makes *seg_name1* a protected-mode alias of *seg_name2* or *group_name*.

- *seg_name1*'s base and limit are set to the base and limit of *seg_name2*.
- *seg_name1* can have its own attributes and selector.
- Any data previously located in *seg_name1* is lost.

*Example*

```
create ldt_alias :: limit=0ffffh
alias ldt_alias :: sys_ldt0
ldt sys_ldt0[1] :: ldt_alias
```

*Segment-modification command*

## ATTRIBUTE *seg_list* :: *attribute_list*

Use this command to alter the attributes of a protected-mode segment.

- Only the attributes in *attribute_list* are changed. All other descriptor fields are left intact.
- Use the items in table 5.3 just below to create an *attribute_list*. Items may be used repeatedly and in any order. Separate entries with commas or spaces.

*Table 5.3. Segment attributes*

| Attribute | Descriptor correspondence |
|---|---|
| DPL=*number* | Set the privilege level in descriptor for segment |
| LIMIT=*number* | Set segment limit |
| LIMIT+=*number* | Increase current limit |
| BYTEGRAIN \| PAGEGRAIN | Byte or page granularity used for limit in descriptor |
| PRESENT \| NOTPRESENT | Present bit in descriptor |
| AVAILABLE \| NOTAVAILABLE | Available bit in descriptor |
| USE32 \| USE16 | 16- or 32-bit segment |
| RO \| RW \| ROED \| RWED \| EO \| ER \| CEO \| CER | Set segment type in descriptor |
| PAGE.PRESENT \| PAGE.NOTPRESENT | Page-present bit |
| PAGE.RO \| PAGE.RW | Read only or Read Write page |
| PAGE.USER \| PAGE.SUPER | User or supervisor protection level |
| PAGE.ACCESSED \| PAGE.NOTACCESSED | Page accessed bit |
| PAGE.DIRTY \| PAGE.NOTDIRTY | Page dirty bit |

***Segment-type abbreviations***

| | |
|---|---|
| RO | Read only, data segment |
| RW | Read/Write, data segment |
| ROED | Read only/Expand down, data segment |
| RWED | Read/Write/Expand down, data segment |
| EO | Execute only, code segment |
| ER | Execute/Read, code segment |
| CEO | Execute only/Conforming, code segment |
| CER | Execute/Read/Conforming, code segment |


You cannot use the LIMIT attribute to decrease the size of a segment that is created by your application.

***Example***

```
attribute init_text :: limit=1000H,
+                      dpl=0,
+                      present
```

*Segment-location command*

## BASE *seg_list* :: *addressL*

This command forces the descriptor base for a segment or group to be the linear address given.

- The segment's physical and linear addresses are not affected, but the offset of its logical address is shifted.
- This is typically used with flat-model applications to make the offset into a segment match its physical address.

*Example*

```
base init_text :: 4000L
```

## BIN[ARY] [*filename*]

*Output command*

This command creates a .BIN binary output file and can also be used to change the name of the file, which by default is the same name as the .CMD file that is the CSi-Locate input command file.

- By default, no absolute, binary, or hex file is output. You must use one of the three output commands (**ABSOLUTE**, **BINARY**, or **HEX**) to generate output.
- These output commands can be used one at a time, all together, or in any combination. Each command will generate one output file.

*Examples*

```
bin test_app.bin
binary test_app
bin csamp.bnr
```

*Protected-mode segment command*

## CALL286 *seg_list* :: *gate_options*
## CALL386 *seg_list* :: *gate_options*

These commands create segments for call-gate descriptors that can be accessed symbolically.

- Use the items in table 5.4, "Gate Attributes," just below to form your *gate_options*. Items may be used in any order and may be repeated, separated by a space or comma.
- DPL and COUNT both default to zero.
- PRESENT | NOTPRESENT defaults to PRESENT.
- ENTRY defaults to the address stored at offset 0 within segment *seg_name*. You can use support macros to predefine these values in your assembly module. See the macro file CSILOC.INC, which is located in the directory where you installed CSi-Locate (see p. 4 for a list of the macros).

*Table 5.4. Gate attributes*

| Gate attributes | Descriptor correspondence |
|---|---|
| DPL=*number* | DPL bits |
| COUNT=*number* | Word count for call gate |
| PRESENT \| NOTPRESENT | Present bit |
| ENTRY=*ptr_value* | Code location gate vectors to |

*Example*

The following example builds a 386 call-gate descriptor named **sys_init**, and sets its address (selector:offset) to the logical address of the public symbol **init_regs** with the following attributes:

DPL=2
Word count field=3
Marked present

```
call386 sys_init :: dpl=2 count=3 present entry=init_regs
```

## CPU *cpu_name*

*Input command*

This command is used to specify the exact processor of the target.

- This command must be placed near the beginning of the command file, before any segment location or manipulation commands.
- If this command is omitted, CSi-Locate defaults to the following processors (CSi-Locate can determine from the executable file what the application's mode is):

| | |
|---|---|
| for a real-mode application | 8086 |
| for a protected-mode application | 80386 |

- Use the terms in table 5.5 just below to specify *cpu_name.*

*Table. 5.5. CPU names*

| | | | | | |
|---|---|---|---|---|---|
| Pentium | | | | | |
| 486 | 486SX | 486DX | | | |
| 386 | 386SX | 386DX | 386EX | | |
| 376 | | | | | |
| 286 | | | | | |
| 188 | C188 | 188EA | 188EB | 188EC | 188XL |
| 186 | C186 | 186EA | 186EB | 186EC | 186XL |
| 88 | | | | | |
| 86 | | | | | |
| V20 | V30 | V40 | V50 | | |

*Examples*

```
cpu pentium
cpu C186
```

***Segment-creation command***

## CREATE *seg_name* [:: *attribute_list*]

This command creates a segment with the given name and optional attributes.

- *Seg_name* must not conflict with any name of a segment already defined by the application.
- *Attribute_list* may contain any of the attributes in table 5.4 on p. 66.
- Segments are placed in memory in the order in which they are created unless they are explicitly located otherwise by the user.

***Example***

```
create csimon_rom :: limit=2000H
```

## DEBUG [*filename*]

*Output command*

DEBUG controls the generation of symbolic information. It has the following characteristics:

- If you want symbolics, you must use this command. By default, CSi-Locate does not generate symbolics.
- **DEBUG** generates a .BUG (*APPLICATION*.BUG) file for real-mode applications. The .ABS file contains no debug information.
- If a *filename* is used, the .BUG file generated will have the specified name rather than the default name, which is the same as the command file's name.
- For protected-mode applications, symbolics are placed in the absolute output file produced by CSi-Locate.

*Examples*

```
debug
debug c:\embed\new_app\prog1.bug
```

*Input command*

## EXEC *filename*

This command is used to change the default file name and extension of the executable file that is to be input to CSi-Locate. The default filename is the same as the input .CMD file's name, and the default extension is .EXE. **MAP** is the other "input-file command."

- **EXEC** changes the default executable input file.
- **MAP** changes the default input map file.
- Both of these input-file commands must be placed at the beginning of the command file, before any segment manipulation and location commands.

*Examples*

```
exec csamp.exe
exec c;\csiloc\csamp.exp
```

## FIXUP *kind ptr_value* = (*ptr_value* | *num_value* | *seg_value*)

*Segment-modification command*

This command allows you to make simple modifications to your application while using CSi-Locate.

*Example*

- When using the startup code supplied with Soft-Scope, the label **cs_dgroup** must be zeroed for your application to build. Use the following to change the value of **cs_dgroup**:

```
fixup word cs_dgroup = group dgroup
```

- If you are not using Soft-Scope startup code, use **FIXUP** to change the values of the symbols to set up your stack.
- Use table 5.6 for values for *kind*:

*Table 5.6 Fixup kinds and their byte sizes*

| KIND | #BYTES | KIND | #BYTES |
|---|---|---|---|
| BYTE | 1 | NEAR16 | 2 |
| DWORD | 4 | NEAR32 | 4 |
| FAR16 | 4 | PHYSICAL | 4 |
| FAR32 | 6 | SELECTOR | 2 |
| LIMIT16 | 2 | TABLE | 6 |
| LIMIT32 | 4 | WORD | 2 |
| LINEAR | 4 | | |

*Example*

- Here's a simple example taken from an actual .CM file. It creates a segment **startjmp,** fixes it up so that it contains a hard-coded jump to **main** (**0xea** is the opcode for JMP), and then locates it at the address of the hardware reset.

```
[20] verbose on
[21] create startjmp :: limit = 5
    >>>  Created segment: STARTJMP.
    >>>  LIMIT=00000005 for STARTJMP
[22] fixup byte startjmp:0 = 0xea
[23] fixup far16 startjmp:1 = main
[24] locate startjmp :: 0FFFF0p
    >>>  STARTJMP located at 000FFFF0L.
```

*Table-constructor command*

## GDT *seg_name* [*range*] [:: (RESERVE | *seg_list*)]

This command is used to build the protected-mode GDT table.

- *seg_name* is where the table will be placed, and must be defined in your application (one way to do this is to use the macro **def_tbl**, mentioned on p. 4, in your startup code) or created with the locator command **CREATE**.

  If the only parameter used is *seg_name*, an empty table is created except for the default null and alias slots.

- *range* specifies the starting and optional ending index. The example below uses *range* to reserve slots for the CSi-Mon monitor:

*Example*

```
gdt sys_gdt[3..64] :: reserve
```

- RESERVE reserves the specified slots for system, monitor, or other uses. These slots are set to zero.
- When a range is not specified for the GDT command, the default starting slot is 3. GDT[0] is null, GDT[1] is the GDT alias, and GDT[2] is the IDT alias.
- Not all gates can be placed in all tables. Only the following can be placed in the GDT table:

  286/386 call gates
  Task gates

*Example*

The following example places the first segment found in the input map file at slot 5 of **tmp_gdt**. All other segments are placed in default order starting at slot 6:

```
gdt tmp_gdt[5] :: *
```

## HEX [*filename*]

*Output command*

This command creates a .HEX output file and can also be used to change the name of the file, which by default is the same name as the .CMD file that is the CSi-Locate input command file.

- By default, no absolute, binary, or hex file is output. You must use one of the three output commands (**ABSOLUTE**, **BINARY**, or **HEX**) to generate output.
- These output commands can be used one at a time, all together, or in any combination. Each command will generate one output file.
- The output format of hex files is a function of the **CPU** command and the executable according to the following conditions:

| | |
|---|---|
| Intel 32-bit hex: | CPU >= 386 |
| Intel absolute 8086 hex: | CPU < 386 and real mode |

*Examples*

```
hex prom.hex
hex c:\newapp\eprom
```

*Table-constructor command*

## IDT *seg_name* [*range*] [:: (RESERVE | *seg_list* )]

This command is used to build the protected-mode IDT table.

- *seg_name* is where the table will be placed, and must be defined in your application (one way to do this is to use the macro **def_tbl**, mentioned on p. 4, in your startup code) or created with the locator **CREATE** command.

  If the only parameter used with these commands is *seg_name*, an empty table is created except for the default null and alias slots.

*Example*

- *range* specifies the starting and optional ending index. The example below uses *range* to reserve slots for the CSi-Mon monitor:

```
idt sys_idt[0..40] :: reserve
```

- RESERVE reserves the specified slots for system, monitor, or other uses. These slots are set to zero.
- GDT[2] is the IDT alias.
- Not all gates can be placed in all tables. The following are the gates that can be put in the IDT table:

  286/386 trap gates
  286/386 interrupt gates
  Task gates

*Example*

```
create int_114
int386 int_114 :: entry=timer_interrupt, DPL=0
IDT sys_idt[41] :: int_114
```

## INIT16R *seg_name* :: *seg_list*
## INIT16P *seg_name* :: *seg_list*
## INIT32P *seg_name* :: *seg_list*

*Segment-definition command*

These commands compress data from the segments in *seg_list*, which are to be located in RAM, and store the compressed data in the ROM segment *seg_name*. Use these commands when you have constants or data that you want located in RAM and that need to be initialized at boot-up time.

- **INIT16R** applies to 16-bit real-mode applications.
- **INIT16P** applies to 16-bit protected-mode applications.
- **INIT32P** applies to 32-bit protected-mode applications.
- The data is stored in a compressed form in ROM.
- The macros **raminit_16r**, **raminit_16p**, and **raminit_32p**, which can be found in the file CSILOC\CSILOC.INC, unpack the data and copy it into the RAM segments in *seg_list*.

*Example*

The following example compresses the data in all of the segments in class **data** and the segment **const** and stores the compressed data in the segment **ram_init**. You can use the macros that are mentioned just above (in the last bulleted item) in your startup code to unpack and copy the data back to class **data** and segment **const**.

```
init16r ram_init :: class data segment const
```


If you are building an application with a Microsoft compiler, it places data into a data segment even if your code does not. You should initialize the data in this data segment with one of the **INIT** commands described on this page.

*Protected-mode segment command*

## INT286 *seg_list* :: *gate_options*
## INT386 *seg_list* :: *gate_options*

These commands create segments for interrupt-gate descriptors that can be accessed symbolically.

- They operate just like the **CALL** commands described above on p. 66 with the following exception:
- The option COUNT is not used.

*Example*

```
int286 int_gate :: dpl=0, present
```

## INTEGRITY *seg_list*

***Segment-location command***

This command forces CSi-Locate to locate and include as part of your application any empty spaces or padding within a segment. This is quite helpful if your compiler writes extraneous data into segments that CSi-Locate isn't otherwise aware of.

- The effect of this command is to preserve any existing "padding" in segments.
- If you don't use this command, there are cases in which the locator suppresses a certain amount of padding when locating a segment.
- Empty space can occur, for example, in the segments that you declare for the GDT and IDT; use of this command would preserve all of that space.

***Example***

```
integrity *
```

*Table-constructor command*

## LDT *seg_name* [*range*] [:: (RESERVE | *seg_list* )]

This command is used to build the protected-mode LDT table.

- *seg_name* is where the table will be placed, and must be defined in your application (one way to do this is to use the macro **def_tbl**, mentioned on p. 4, in your startup code) or created with the locator command **CREATE**.

  If the only parameter used with these commands is *seg_name*, an empty table is created except for the default null and alias slots.

***Example***

- *range* specifies the starting and optional ending index. The example below uses *range* to reserve the first ten slots:

```
ldt sys_ldt[0..9] :: reserve
```

- RESERVE reserves the specified slots for system, monitor, or other uses. These slots are set to zero.

***Example***

- When a range is not specified for the LDT command, the default starting slot is LDT[2]. LDT[0] is null, and LDT[1] is its alias, which you need to set up in a way like the following:

```
create ldt_alias limit=0ffffh
alias ldt_alias :: sys_ldt0
ldt sys_ldt0[1] :: ldt_alias
```

- Not all gates can be placed in all tables. The following are the gates that can be put in the LDT table:

  286/386 call gates

***Example***

The following example places class code in **ldt_1** starting at slot 2, then places all segments except those in **class code** into **ldt_2,** starting at slot 2:

```
ldt ldt_1[2] :: class code
ldt ldt_2 :: * except class code
```

## LOCATE *seg_list* :: (*addressL* | *addressP*)

***Segment-location command***

This command locates segments, classes, or groups in ROM or RAM, beginning at the given address.

- **LOCATE** assigns an address to the first segment in *seg_list*. If a class is given, the address is assigned to the first segment in the class.
- Once a segment is located, its location is permanent.
- Multiple instances of this command can be used in a command file, locating different segments, groups, or classes..
- Individual segments in groups cannot be located without the rest of the group.
- Individual segments in classes can be located by themselves.
- *addressL* is a linear address must have an "L" suffix; *addressP* is a physical address must have a "P" suffix;

  **Linear locate** locates groups at specific addresses and maintains segments at adjacent linear addresses.

  **Physical locate** places segments and allows nonadjacent addresses for same-group segments.

***Example***

The following example first locates segment **seg1**, which let us assume is in class **a_class**, at 50000P, then locates the remaining segments in the **a_class** class, and finally locates the segments in **d_class**:

```
locate seg1 class a_class d_class :: 50000P
```

***Example***

The next example uses the EXCEPT keyword to prevent **seg1** from being located with the rest of its class. A separate **LOCATE** command or some other locator command would be needed to locate **seg1**.

```
locate class a_class except seg1 :: 50000P
```


If a segment defined in assembly language gets located a few bytes beyond where you specify with the **LOCATE** command in a .CMD file, define the segment as paragraph aligned in your assembly code to prevent such dislocation.

*Input command*

## MAP *filename*

This command is used to change the default file name and extension of CSi-Locate's input map file. The default filename is the same as the input .CMD file's name, and the default extension is MAP. **EXEC** is the other "input-file command."

- **EXEC** changes the default executable input file.
- **MAP** changes the default map input file.
- Both of these input-file commands must be placed at the beginning of the command file, before any segment manipulation and location commands.

*Example*

```
map csamp.map
```

## PAGEDIRECTORY *dir_name* [*range* ] :: *seg_list*

***Segment-definition command***

This command defines the given segment *dir_name* as a page directory and the segments in *seg_list* as page tables.

- You must create the segment *dir_name* with the **CREATE** command or in your startup code (see p. 4 for the mention of a macro that can be used to create this segment) before you use this command.
- It allocates the exact position of each page table within the page directory.
- The full range of linear addresses used by the application must be accounted for.

For more information, see next page's discussion about **PAGETABLE**.

***Example***

```
pagedirectory dir_name[0] :: page_table
```

*Table-constructor command*

## PAGETABLE *dir_name* :: *seg_list*

This command specifies that the segments in *seg_list* are to be mapped through the page directory *dir_name*.

The **PAGEDIRECTORY** *range* parameter defines the pagetable range, as in table[1], table[2], table[3]..., and is useful when you want to split your application into separate pieces, or if your application is large.

This is because the page table and the linear address of a segment are directly related—given a certain linear address, the physical address associated with a segment will be placed in a specific page table. The CPU controls this, and CSi-Locate can't alter it.

However, by controlling where a segment is located you can control to some extent which page table it is associated with. This is important because the CSi-Locate **PAGEDIRECTORY** and **PAGETABLE** commands set up the page tables, and if they don't set up a table that one of your segments is associated with, a fatal error will occur.

For those segments you don't locate explicitly, their location is dependent upon the order in which they were created. So, if one of your segments ends up in the wrong table, you can put it in another table without explicitly locating it by creating it earlier in the file.

*Examples*

The first example below takes advantage of this feature. The range specified in the **PAGEDIRECTORY** command is [0], and as long as the page tables needed are consecutive, CSi-Locate sets them up. However, if you locate your segments so that you have segments associated with table[0], then skip table[1] and have segments associated with table[2], table[2] will not be set up.

In the second example below, because, let us suppose, some of the segments are located at linear addresses 0f0000000L through 0f00000003L, we need page table [960].

```
pagedirectory dir_name[0] :: page_table
pagedirectory dir_name[960] :: page_table1
```

## PMODE *seg_list* | (*pub_sym1* ((to *pub_sym2*) | (LENGTH *number*)))

*Segment-modification command*

**RMODE** and **PMODE** allow mixed-mode applications to be built properly. Use them to change the assumed mode of segments or parts of segments.

- Use **PMODE** *seg_list* to mark an entire segment as protected mode.
- Any segment or segment portion marked as protected mode will reference segments using their protected-mode selectors.
- All public symbols used in this command (as code boundaries for specific purposes) must be in the same segment.

*Examples*

Given the following segments

```
DSEG -- real-mode segment
PSEG -- protected-mode segment
CODE_REAL -- real-mode segment
```

with the following real-mode assembly code,

```
public prot_start
public prot_end

CODE_REAL segment eo;
                    mov ax, DSEG
                    xor ax, 2
prot_start:
                    mov bx, PSEG
                    mov cx, bx
prot_end:
                    inc bx
end CODE_REAL
```

The example below causes the instructions **mov bx, PSEG** and **mov cx, bx** to have a protected-mode fixup:

```
pmode prot_start to prot_end
```

An alternate method is to use one public symbol to mark the beginning of the section and then to use the LENGTH keyword to specify how long it is. The following marks 10 bytes:

```
pmode prot_start length 10
```

*Output command*

## PRINT

The PRINT command tells CSi-Locate to print public-symbol information to the .CM conversion map file.

- The name and location of each public symbol are listed module by module. Public symbols include symbols declared PUBLIC in assembly files, static C variables, global variables, and names of procedures from user modules and libraries.
- No public symbolic information is put in the conversion map file by default.

*Example*

```
print
```

## RAM *seg_list*

*Segment-modification command*

Use this command to specify segments that you *do not* want placed in the output file.

- All segments not specified with this command will be put in the output files requested by the **ABSOLUTE**, **HEX**, and **BIN** commands.

*Example*

Given the following segments:

```
data_seg, code_seg, stack_seg, temp
```

The following example places all but **data_seg** and **temp** in the output file:

```
ram data_seg temp
```

*Segment-modification command*

## RMODE *seg_list* | (*pub_sym1* ((to *pub_sym2* ) | (LENGTH *number* )))

**RMODE** and **PMODE** allow mixed-mode applications to be built properly. Use them to change the assumed mode of segments or parts of segments.

- Use **RMODE** *seg_list* to mark an entire segment as real mode.
- Any segment or segment portion marked as real mode will reference segments using their real-mode selectors.
- All public symbols used in this command must be in the same segment.

See the example use of **PMODE** on p. 83 above.

## ROMBASE *seg_list* :: *addressP*

*Segment-modification command*

This command allows you to decrease hex-record addresses to set the base address of ROM.

- Use this command if you are burning your application into ROM and your ROM programmer doesn't allow you to set the ROM base address.

*Example*

```
rombase init_text :: 4000P
```

*Segment-modification command*

## ROMMOVE *seg_list* :: *addressP*

This command allows you to increase hex-record addresses.

- Use this command if you want to locate records out of RAM in ROM or locate records to a higher address entirely within RAM or ROM.

*Example*

```
rommove init_tex :: 2000P
```

## TASKGATE *seg_list* :: *gate_options*

*Protected-mode segment command*

This command sets up task-gate descriptors and operates like the **CALL** commands on p. 66 above, with the following exceptions:

- The entry point must be a segment previously defined as a TSS.
- The option COUNT is not used.

*Example*

```
taskgate tss.task1 :: dpl=0 present
```

*Protected-mode segment command*

## TRAP286 *seg_list* :: *gate_options*
## TRAP386 *seg_list* :: *gate_options*

These commands set up trap-gate descriptors and operate like the **CALL** commands shown above on p. 66, with the following exception:

- The option COUNT is not used.

*Example*

- When you define a trap gate using **TRAP286** or **TRAP386**, you must include a command to place the gate in the IDT. The following example creates a segment, defines it as a trap gate, then places it in slot 50 of the IDT (this assumes that the segment **idtsys** has already been created):

```
create trap_gt
trap286 trap_gt :: dpl=1 present
+                  entry=init_text
idt idtsys[50]  :: trap_gt
```

## TSS286 *seg_list* :: *assign_list*
## TSS386 *seg_list* :: *assign_list*

***Protected-mode segment command***

These commands set segments in *seg_list* as TSS segments and allow you to specify TSS fields using *assign_list*.

- 16-bit segments may be defined in your application. 32-bit segments must be created with the **CREATE** command.
- An initial TSS is created only for protected-mode applications, and only when one of the commands above is used. The first TSS defined in the command file is the initial TSS.
- All fields not explicitly set are left intact.
- Use the SEGMENTOF and OFFSETOF keywords to specify what part of a public symbol's address to use. See pp. 56–57 above for command syntax elements.

TSS descriptors can only be placed in the GDT. Attempting to place them in the IDT or LDT results in an error.


***Examples***

The following example builds a TSS called **tss_new**. CS:EIP, DS, and FL.IF are explicitly set, while all other fields are left unchanged (note the use of the line-continuation character at the beginning of the second line):

```
tss386 tss_new :: cs:eip=main, ds=data_seg,
+                 fl.if=0x1
```

The next example builds an initial TSS, sets the CS:EIP, and sets fields in two other TSS segments as well:

```
tss386 tss_init :: cs:eip=init_code
tss386 task_1 :: ax=2
tss386 task_2 :: efl.if=1
```

See tables 5.7 and 5.8 on the following pages for applicable TSS fields.

***Table 5.7.***
***TSS286 fields***

AX=*number* BX=*number* CX=*number* DX=*number*
SI=*number* DI=*number* BP=*number*
DS=(*number* | *seg_name* | *pub_sym*)
ES=(*number* | *seg_name* | *pub_sym*)
CS:IP=(*number:number* | *seg_name* | *pub_sym*)
    CS=(*number* | *seg_name* | *pub_sym*)
    IP=*number*
SS:SP=(*number:number* | *seg_name* | *pub_sym*)
    SS=(*number* | *seg_name* | *pub_sym*)
    SP=*number*
SS0:SP0=(*number:number* | *seg_name* | *pub_sym*)
    SS0=(*number* | *seg_name* | *pub_sym*)
    SP0=*number*
SS1:SP1=(*number:number* | *seg_name* | *pub_sym*)
    SS1=(*number* | *seg_name* | *pub_sym*)
    SP1=*number*
SS2:SP2=(*number:number* | *seg_name* | *pub_sym*)
    SS2=(*number* | *seg_name* | *pub_sym*)
    SP2=*number*
LDTR=(*number* | *seg_name* | *pub_sym*)
LINK=(*number* | *seg_name* | *pub_sym*)
FL=*number*
    FL.NT=*number* FL.IOPL=*number*
    FL.OF=*number* FL.DF=*number* FL.IF=*number*
    FL.TF=*number* FL.SF=*number* FL.ZF=*number*
    FL.AF=*number* FL.PF=*number* FL.CF=*number*

EAX=*number* EBX=*number* ECX=*number* EDX=*number*
ESI=*number* EDI=*number* EBP=*number*
DS=(*number* | *seg_name* | *pub_sym*)
ES=(*number* | *seg_name* | *pub_sym*)
FG=(*number* | *seg_name* | *pub_sym*)
GS=(*number* | *seg_name* | *pub_sym*)
CS:EIP=(*number:number* | *seg_name* | *pub_sym*)
CS=(*number* | *seg_name* | *pub_sym*)
EIP=*number*
SS:ESP=(*number:number* | *seg_name* | *pub_sym*)
SS=(*number* | *seg_name* | *pub_sym*)
ESP=*number*
SS0:ESP0=(*number:number* | *seg_name* | *pub_sym*)
SS0=(*number* | *seg_name* | *pub_sym*)
ESP0=*number*
SS1:ESP1=(*number:number* | *seg_name* | *pub_sym*)
SS1=(*number* | *seg_name* | *pub_sym*)
ESP1=*number*
SS2:ESP2=(*number:number* | *seg_name* | *pub_sym*)
SS2=(*number* | *seg_name* | *pub_sym*)
ESP2=*number*
LDTR=(*number* | *seg_name* | *pub_sym*)
LINK=(*number* | *seg_name* | *pub_sym*)
EFL=*number*
EFL.NT=*number* EFL.IOPL=*number*
EFL.OF=*number* EFL.DF=*number* EFL.IF=*number*
EFL.TF=*number* EFL.SF=*number* EFL.ZF=*number*
EFL.AF=*number* EFL.PF=*number* EFL.CF=*number*
EFL.RF=*number* EFL.*VM*=*number* EFL.CF=*number*
CR3=*number*
CR3.PDBR=*number* CR3.PCD=*number* CR3.PWT=*number*
IO_MAP=*number* TRAP=*number*

***Table 5.8. TSS386 fields***

*Locator-debugging command*

## VERBOSE (ON | OFF)

You can use this command to print additional information to the conversion map file.

- The default is **VERBOSE OFF**.
- When ON, CSi-Locate prints detailed information to the conversion map file as each command executes. The information printed depends on the command. For example, if the command just executed impacts segment location, detailed information about where and how the segment was located is placed in the conversion map file.
- **VERBOSE** can be used anywhere in the command file, and can be turned on or off as often as you wish in the same file.
- **VERBOSE** can be used anywhere in the command file, and can be turned on or off as often as you wish in the same file.

The following exemplifies the sort of output provided by **VERBOSE ON**

*Example*

```
[20] verbose on
[21] create startjmp :: limit = 5
    >>>  Created segment: STARTJMP.
    >>>  LIMIT=00000005 for STARTJMP
[22] fixup byte startjmp:0 = 0xea
[23] fixup far16 startjmp:1 = main
[24] locate startjmp :: 0FFFF0p
    >>>  STARTJMP located at 000FFFF0L.
```

# Appendix A Error and Warning Messages

A

CSi-Locate generates messages when it cannot execute a command or process your application as specified.

There are five kinds of messages, organized in this chapter as follows:

| | | |
|---|---|---|
| 1. | Fatal errors | Processing halts immediately—no output files are generated. |
| 2. | System errors | Processing halts immediately—no output files are generated. |
| 3. | Errors | Processing continues—no output files are generated. |
| 4. | Syntax errors | Processing halts after command file has been interpreted—no output files are generated. |
| 5. | Warnings | Processing continues until completed—output files are generated. |

Where possible, messages are listed in the following format:

1. ***** message** or **<message>**
2. Explanation that describes why the message was displayed
3. What to do to eliminate the problem here or avoid it in the future

## Table of Contents

***** Fatal: Corrupted map listing file.**

General error message.

Try recreating the map file with the linker.

**< System - No such file or directory: "*filename*." >**

The operating system can't find the file or directory specified.

Check to make sure the path and filename are correct.

**< System - Permission denied: "*filename*." >**

*Filename* may be write protected.

**< System - Line too long: "*filename*." >**

CSi-Locate encountered a line in *filename* that contains more than 222 characters.

Change the line length.

***** ERROR: Attempting to locate library module.**

CSi-Locate does not support Windows library modules.

***** ERROR: Bad fixup, *segment:offset* does not exist in file *filename*.**

This means that a fixup was read out of the executable but the address to be fixed up does not exist in the executable space.

Try regenerating the executable.

***** ERROR: Bad fixup segment reference, *segment:offset*.**

The address contained at a fixup location points to a nonexistent *segment:offset*.

Regenerate the executable.

***** ERROR: Call gates and TSS descriptors are not allowed in IDT tables.**
***** Line *#line_number*.**

***** ERROR: Cannot ROM all of segment *segment_name*.**
***** Address *address* is over 0fffffH (1MB).**

The protected-mode segment being saved has an address greater than 1 MB.

***** ERROR: Cannot ROM all of segment *segment_name*.**
***** Address *address* is over 0ffffffH (16 MB).**

The protected-mode segment being saved has an address greater than 16 MB.

***** ERROR: Corrupted fixup chain at *segment:offset*.**

A particular fixup references a location past the end of the segment, or the fixup is a duplicate fixup.

Regenerate the executable file.

***** ERROR: Duplicate gate name in gate *gate_name*, gate not built.**
***** Line *#line_number*.**

Gate names must be unique when specifying them with the GATE command.

Rename the gate in question.

***** ERROR: First segment contains loader.**

Your executable code contains Windows structures that CSi-Locate does not support.

***** ERROR: Fixup at *selector:offset* is in an unknown segment.**

While processing a fixup from the executable, CSi-Locate could not determine which segment the fixup was to be applied to.

In most cases this indicates a corrupted executable.

***** ERROR: Fixup at *selector:offset* references an unknown segment *segment_name*.**

While processing a fixup from the executable, CSi-Locate could not determine which segment a fixup was referencing. In most cases this indicates a corrupted executable.

***** ERROR: Gate *gate_name* was previously assigned to GDT[*index*].**
***** Line *#line_number*.**

***** ERROR: Gates cannot alias other segments.**
***** Line *#line_number*.**

***** ERROR: Gates cannot be aliased.**
***** Line *#line_number*.**

Gate descriptors cannot have other segments alias them.

***** ERROR: Index *table_name*[*index*] is reserved.**
***** Line *#line_number*.**

***** ERROR: Interrupt/Trap gates are not allowed in GDT or LDT tables.**
***** Line *#line_number*.**

***** ERROR: Protected mode tables cannot be in groups.**
***** Line *#line_number*.**

***** ERROR: Protected mode tables must be paragraph aligned.**
***** Line *#line_number*.**

***** ERROR: Public symbol *string* is not in the same**
***** segment as public symbol *string*.**
***** Line *#line_number***

CSi-Locate cannot find a public symbol in the public symbol table that is given in the command file.

All public symbols used within a single RMODE or PMODE command must be in the same segment.

***** ERROR: Public symbol *string* not found.**
***** Line *#line_number*.**

CSi-Locate cannot find a public symbol in the public symbol table that is given in the command file.

Look in the map file to track down the problem. The symbol may have a '_' appended to it, it may be spelled incorrectly, the letters may be the wrong case, or it may be missing altogether from the map file.

***** ERROR: Range goes beyond limit of the table.**
***** Line *#line_number*.**

A given GDT/IDT/LDT range exceeds the maximum number of slots available.

Either make the table larger or change the range given in the command file.

***** ERROR: Real-mode reference to segment is not possible.**

A real-mode segment is referencing a protected-mode segment.

Use the PMODE command to fix the segment.

***** ERROR: *Segment_name* was previously assigned to GDT[*index*].**
***** Line *#line_number*.**

```
*** ERROR: Segment_name was previously assigned to LDT[index].
***        Line #line_number.
```

```
*** ERROR: Segment_name was previously assigned to LDT table_name[index]
***        but table_name[index] is used.
***        Line #line_number.
```

```
*** ERROR: Segment segment_name has not been placed in a table.
***        Line #line_number.
```

CSi-Locate is trying to build a protected-mode application, but can't find any tables to place segments in.

Use the IDT, GDT, and LDT commands to build protected-mode tables.

```
*** ERROR: Segment segment_name is already defined as a 286 TSS.
***        Line #line_number.
```

```
*** ERROR: Segment segment_name is already defined as a 386 TSS.
***        Line #line_number.
```

```
*** ERROR: Segment segment_name is already defined as a gate.
***         Line #line_number.
```

```
*** ERROR: Segment segment_name is already defined as an LDT.
***         Line #line_number.
```

```
*** ERROR: Segment segment_name is already defined as the GDT.
***        Line #line_number.
```

```
*** ERROR: Segment segment_name is already defined as the IDT.
***        Line #line_number.
```

```
*** ERROR: Segment segment_name is an alias and cannot be located.
***        Line #line_number.


*** ERROR: Segment segment_name1 is in the same segment
***        as segment segment_name2.  Cannot create the alias.
***        Line #line_number.


*** ERROR: Segment segment_name is in a group and cannot be moved.
***        Line #line_number.


*** ERROR: Segment segment_name shares a selector with segment segment_name.
***        Class class_name can't be moved.
***        Line #line_number.


*** ERROR: Segment segment_name shares a selector with segment segment_name.
***        Group group_name can't be moved.
***        Line #line_number.


*** ERROR: Segment segment_name shares a selector with segment segment_name.
***        Segment segment_name can't be moved.
***        Line #line_number.


*** ERROR: Slot[index] has a fixup referencing it,
***        and cannot make an assignment.
***        Line #line_number.
```

Your code references the given table slot.

Remove the reference from your code or force a different slot assignment in the command file.

***** ERROR: *String* is an invalid file type.**

An unrecognized output file type has been requested.

Replace with the proper file type keyword.

***** ERROR: *Table_name segment_name* is full.**
***** No selector assigned for *segment_name*.**
***** Line *#line_number*.**

To correct, make the table larger by increasing the defined size in the assembly module.

***** ERROR: Table *table_name* cannot be a real-mode segment.**

All tables must be protected-mode segments.

***** ERROR: Task gates and TSS descriptors are not allowed in LDT tables.**
***** Line *#line_number*.**

***** ERROR: There is a directory named *directory_name*.**
***** Append '.cmd' to command file name.**

The directory containing your command file has a subdirectory with the same name as the command file.

Either rename the directory, rename the command file, or append .CMD to the command file name when invoking the locator.

***** ERROR: TSS *segment_name* too small, must be at least 44 bytes.**
***** Line *#line_number*.**

***** ERROR: TSS *segment_name* too small, must be at least 104 bytes.**
***** Line *#line_number*.**

```
*** ERROR: TSS tables cannot be in groups.
***        Line #line_number.
```

```
*** ERROR: TSS tables must be paragraph aligned.
***        Line #line_number.
```

```
*** ERROR: Unable to initialize register_name register, segment:offset.
```

The initial segment value read from the executable for a particular register is invalid, i.e., it references a nonexistent segment.

Regenerate executable.

```
*** SYNTAX: Class class_name is an invalid class name.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected a seg_list or pub_sym.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected a TSS attribute list.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected attribute_list.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected colon in segment:offset expression.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected gate_attr_list.
***         Line #line_number.
```

```
*** SYNTAX: CSi-Locate expected TO or LENGTH keyword.
***         Line #line_number.


*** SYNTAX: Expected a '::' separator.
***         Line #line_number.


*** SYNTAX: Expected a class name.
***         Line #line_number.


*** SYNTAX: Expected a cpu name.
***         Line #line_number.


*** SYNTAX: Expected a filename.
***         Line #line_number.


*** SYNTAX: Expected a gate name or '::'.
***         Line #line_number.


*** SYNTAX: Expected a group name.
***         Line #line_number.


*** SYNTAX: Expected a hex address.
***         Line #line_number.


*** SYNTAX: Expected a segment name.
***         Line #line_number.


*** SYNTAX: Expected an '=' in the attribute list parameter.
***         Line #line_number.
```

```
*** SYNTAX: Expected a Group, Class, or Segment keyword.
***         Line #line_number.


*** SYNTAX: Expected Intel hex file type.
***         Line #line_number.


*** SYNTAX: Expected ON or OFF switch.
***         Line #line_number.


*** SYNTAX: Group group_name is an invalid group name.
***         Line #line_number.


*** SYNTAX: Invalid decimal number.
***         Line #line_number.


*** SYNTAX: Invalid hex number.
***         Line #line_number.


*** SYNTAX: Invalid number format.
***         Line #line_number.


*** SYNTAX: Invalid range parameter.
***         Line #line_number.


*** SYNTAX: Limit is set too small.  It must be at least number.
***         Line #line_number.


*** SYNTAX: Seg_gate_list parameter expected.
***         Line #line_number.
```

*** SYNTAX: *Seg_list* parameter expected.
***  Line *#line_number*.

*** SYNTAX: Segment *segment_name* is an invalid segment name.
***  Line *#line_number*.

*** SYNTAX: *String* is an invalid *attribute_list* parameter.
***  Line *#line_number*.

*** SYNTAX: *String* is an invalid cpu name.
***  Line *#line_number*.

*** SYNTAX: *String* is an invalid *gate_attr_list* parameter.
***  Line *#line_number*.

*** SYNTAX: *String* is an invalid hex address.
***  Line *#line_number*.

***SYNTAX: *String* is an invalid ON or OFF switch parameter.
***  Line *#line_number*.

***SYNTAX: *String* is an invalid output file type.
***  Line *#line_number*.

***SYNTAX: *String* is an invalid TSS286 field.
***  Line *#line_number*.

***SYNTAX: *String* is an invalid TSS386 field.
***  Line *#line_number*.

```
***SYNTAX: String is not a command.
***        Line #line_number.
```

```
*** WARNING:Class class_name has been previously located.
***         Line line_number.
```

```
*** WARNING:Constants not supported,
***         :module_name.const_name discarded.
```

C-type constants are not supported.

```
*** WARNING:DPL parameter greater than 3, defaulting to 3.
```

```
*** WARNING:Duplicate LDT segments/gates must have the same selector,
***         using LDT segment_name.
***         Line line_number.
```

An attempt was made to reassign a segment/gate to a different LDT slot.

The reassignment will not take place; the original assignment will remain intact.

```
*** WARNING:EXEC must come before any location commands.
```

```
*** WARNING:Group group_name has been previously located.
***         Line line_number.
```

```
*** WARNING: Group names not in map listing, defaults used.
```

Each map file has a section where it displays group information.

Set the proper linker switches to correct this.

***** WARNING:Initial RR=*segment* is not a known segment, 0000 used.**

The initial value of the given segment register (RR) does not match any of the executables segments.

***** WARNING:Invalid TYPEDEF:*string*.**
***** Null used.**

Call Concurrent Sciences, inc.

***** WARNING:Limit over 16 Meg for segment *segment_name*,**
***** page granularity used.**
***** Line *line_number*.**

An attempt was made to set the limit field of a descriptor whose granularity was set to byte length.

CSi-Locate changes the descriptors granularity to page granularity and divides the limit 4096 to get the correct limit.

***** WARNING:Line :*modulename_linenumber* specified multiple times,**
***** record ignored.**

When the executable defines a single line number multiple times, CSi-Locate ignores the second definition and processing continues.

***** WARNING:Linker detected errors.**

There are errors, detected at link time, that remain.

Look in the map file for the error explanations, correct them, then relink the application.

***** WARNING:MAP must come before any location commands.**

***** WARNING:New limit must be at least *min_limit*.**
***** Line *line_number*.**

The limit must be at least as long as the segment length.

***** WARNING:No debug symbol records in input file.**

The input executable contained no debug records. The output is still usable, but Soft-Scope will not be capable of symbolic debugging.

***** WARNING:No logical address for :*module_name.symbol_name*,**
***** Symbol discarded.**

Symbols at absolute addresses are not yet supported.

***** WARNING:No logical address for *symbol_name*, symbol discarded.**

Symbols at absolute addresses are not yet supported.

***** WARNING:No public symbols in map listing.**

Public symbols are not required in the map list, however, if you wish to use them, you must set the appropriate linker switches to include public symbols.

***** WARNING:No symbolic records present, DEBUG command ineffectual.**

The application was built without symbols, but a request is being made in the command file to supply symbolic information to the locator output files.

Either remove the DEBUG command or recompile using the proper symbolic switches.

***** WARNING:*Number* exceeds maximum possible value of field *field_name*.**
***** Line *line_number*.**

An attempt was made to assign a larger value than a particular TSS or descriptor field could hold.

***** WARNING: Previously defined data in *Table_name* [*index*] will be lost.**

The specified table slot already has data in it.

CSi-Locate discards the previously defined data.

***** WARNING: Previously defined data in *Table_name* [*index*] will be lost.**
***** Line *line_number*.**

The specified table slot already has data in it.

***** WARNING: Segment *segment_name* from executable not found in map listing.**

Your map file is corrupted. CSi-Locate found a segment defined in the executable but not in the map file.

Rebuild your application.

***** WARNING: Segment *segment_name* has been previously located.**
***** Line *line_number*.**

***** WARNING: Segment *segment_name* has not been located.**
***** Line *line_number*.**

***** WARNING: Segment *segment_name* is already an alias for segment *segment_name*.**

***** WARNING: Segment *segment_name* is full, segment *segment_name* can't fit.**

Make segment #1 larger or copy segment #2 to a different ROM segment.

***** WARNING: Segment *segment_name* is an alias, cannot define as a TSS.**
***** Line *line_number*.**

***** WARNING: Segment *segment_name* is an alias, cannot change to real mode.**
***** Line *line_number*.**

***** WARNING: Selector *hex_number* is not a known segment, 0000 used.**

Some symbolic record references selector *hex_number*, but this selector is not a known segment.

Make segment #1 larger or copy segment #2 to a different ROM segment.

***** WARNING: Selector *hex_number* is not a known segment,**
***** 0000 used.**

Some symbolic record references selector *hex_number*, but this selector is not a known segment.

***** WARNING: Symbolic name '*string*' too long,**
***** truncated to '*string*'.**

A symbol name is too long and has been truncated. Soft-Scope will know it by the truncated version.

***** WARNING: Symbols section corrupted for :*module_name*.**

The symbol records for the given module may be invalid.

***** WARNING: *Table_name* [*index* ]is not empty, using *table_name*[*index* ].**
***** Line *line_number*.**

CSi-Locate found data in the specified table and substituted another table.

***** WARNING: The *wordcount* parameter is greater than 31, defaulting to 0.**
***** Line *#line_number*.**

***** WARNING:Translated TYPDEF's exceed 0xFFFF,**
*****  nnnn Types discarded.**

The total number of type records is limited to 64K. When the record count exceeds this amount, CSi-Locate has no choice but to throw out the excess type definitions.

***** WARNING:Translated TYPDEF's exceed 0xFFFF,**
*****  *nnnn* types discarded.**

The total number of type records is limited to 64K. When the record count exceeds this amount, CSi-Locate has no choice but to throw out the excess type definitions.

***** WARNING:Translated TYPDEF's exceed 0xFFFF in :MODULENAME,**
*****  *nnnn* types discarded.**

***** WARNING:Translated TYPDEF's exceed 0xFFFF in :MODULENAME,**
*****  *nnnn* types discarded.**

***** WARNING:*Typedef_error_message*, NULL type used.**

Some problem occurred in translating a type record. Any symbols that reference this type, Soft-Scope reports as NULL typed.

***** WARNING:*Typedef_error_message* in :*module_name*, NULL type used.**

***** WARNING:Unknown fixup type at *segment:offset*.**

An unknown fixup type has been found. Contact Concurrent Sciences (see box on p. ii).

***** WARNING:Unknown register for :*module_name.procedure_name.register_var*,**
*****  symbol discarded.**

A register variable was defined using a code that CSi-Locate does not recognize. This could be a corrupted executable, or a later addition to the object format.

***** WARNING:Unsupported symbolics format.**

CSi-Locate has detected an input symbolic format it does not recognize.

Reconfirm that the tools being used are supported by the locator.

***** WARNING:When attempting to assign a segment to a table slot, CSi-Locate**
***** found the slot already contained data.**

This could be from a previous assignment made in the command file or from data defined in the assembly module where the table is set up.

In this case, the next available table slot is used.

This is the only line of text on this page.

# Appendix B
# ROMming Protected-Mode Applications

B

This appendix provides a sketch of what you need to do to ROM protected-mode applications.

There is an example of a real-mode ROMmed application discussed on pp. 20–23. That example includes .CMD command file, map file, and located-segment map part of the .CM conversion map file. The crucial RAM-initialization command **INIT16R** is used in the command file. Any initialized RAM data for a ROMmed application needs to be taken care of by means of this important command in conjunction with the use of unpacking and copying macros (see p. 4) that are found in CSILOC\CSILOC.INC. Most of the sample .ASM files in the subdirectories of \CSILOC\SAMP illustrate how these macros are used.

# Preparing to ROM a Protected-Mode Application

***Preparing to ROM a protected-mode application***

Two excellent discussions for preparing to ROM your protected-mode application are the following: (1) James L. Turley, *Advanced 80386 Programming Techniques* (Berkeley, CA: Osborne McGraw-Hill, 1988), pp. 414–421 [Unhappily this source is out of print]. (2) *Pentium Processor User's Manual. Volume 3: Architecture and Programming Manual* (Mt. Prospect, IL: Intel Corporation, Literature Sales, Order Number 241430, 1994), pp. 16-13–16-29.

The advice below on preparing to ROM your protected-mode application is based on these two sources.

Bootstrap code at FFFFFFF0H usually contains a NEAR (intrasegmental) JMP instruction to *reset-initialization code* (the NEAR JMP means the code is placed within the last 64KB of the 4 GB address space) that should do the following (see pp. 16-15 of the Pentium manual for an algorithm for this, and pp. 16-17–16-24 for actual assembly code for this):

(a) perform simple diagnostic testing, e.g., checking the 80386's self-test signature in EAX and testing low memory before building descriptor tables there;

(b) set up protected-mode data structures GDT and IDT by copying them from bootstrap ROM into low RAM (load IDTR as late as possible before switching to protected mode; LDTs, TSSs, and paging apparatus can all be set up after the processor is in protected mode);

(c) switch the processor to protected-mode by setting the Protection Enable (PE) bit to 1;

(d) perform all or some of the following recommended actions (Turley, pp. 419–420):

(i) do a NEAR JMP instruction to flush the prefetch queue so that all subsequent instructions are interpreted as protected-mode, not real-mode instructions;

(ii) load all 6 segment registers with protected-mode descriptors as soon as possible;

(iii) initialize SS and ESP;

(iv) do a FAR JMP to initialize CS;

(v) initialize task register TR;

(vi) then initialize LDTR.

(e) jump to startup code that typically does the following:

(i) Declares data and stack segments

(ii) Initializes data, which is not otherwise initialized to zeroes, including stack.

(iii) Unpacks and copies initialized data from ROM to RAM. The **INIT16P/INIT16R/INIT32P** locator commands (see p. 75) set up the ROM segment(s) that are to hold the data that needs to be copied. There are macros in the file CSILOC\CSILOC.INC. that can perform the unpacking and copying of data from ROM to RAM (see p. 4).

(iv) Performs other initializations, e.g., floating-point emulation software.

(v) Calls main.

***Preparing to ROM a real-mode application***

If you are preparing to ROM a 16-bit real-mode application, take a look at the example discussed on pp. 20–23. The example ROMs a 16-bit real-mode application that is to be run on the Intel 386EX board. The primary task that needs doing in setting it up is setting up RAM-initialization code in ROM. You can define the ROM segment to hold the RAM-initialization code as well as the particular data segments or classes that need to be copied into RAM by using the **INIT16R** locator command, as is done in the example.

The example uses the startup code in SAMP\MSC16R\B16RMSC.ASM with modifications to set up the 386EX hardware. The example's .CMD command file is given, followed by a brief explanation of it, and then you will find the segment map of the located code from the .CM conversion map file.

This is the only line of text on this page.

This index is alphabetized word by word, where a word is considered to be any contiguous string of alphabetic characters. Thus, "arr_chg" precedes "array," and "sym.wordsize" precedes "symbol."

## Symbols (ASCII ordered)



## A



## B



## C

## D



## E



## F

## G



## H



## I



## J

## K



## L

## M



## N



## O



## P

## Q

## R

## S



## T



## U

## V



## W



## X

## Y

## Z