WHAT IS CSP?

CSP (CS Preprocessor) uses the Microsoft C or QuickC compiler as a Telix SALT preprocessor, giving the SALT compiler conditional compile, include file, and macro capabilities.

CSP requires Microsoft's QCL.EXE or CL.EXE in one of the directories in your PATH.

WHAT DOES IT DO?

Not a whole lot. It runs a script file (for example FOO.SLT) through the Microsoft C compiler (either QCL or CL) with the /P switch to create an intermediate file. The intermediate file is lightly edited, then given to the SALT compiler (CS.EXE). Line references in CS error messages are changed from the file and line of the intermediate file to the original file and line number.

FILES INCLUDED:

CSP.EXE	SALT "preprocessor".
CSP.DOC	This file.
CSP.WP5	WordPerfect 5.1 version of this file.

HOW DO I USE IT?

The simple syntax is:

```
CSP sourcefile [outputfile]
```

Which will preprocess and compile a SALT source file. If an output name is given CSP uses it to determine output file names, otherwise output files are named after the source file.

The full syntax is:

```
CSP [-Q"CMD"] [-PC] [-D MACRO=VALUE]... [-I DIR] [-b]
   [-sstack] sourcefile [outputfile]
```

switches (which may begin with either '-' or '/') are:

-Q Use CMD as preprocessor.
-P Preprocess only, do not compile. Creates a .SLI file.
-C Tell C preprocessor not to strip comments.
-D Tell C preprocessor to define MACRO as VALUE.

-I Tell C preprocessor to search DIRECTORY for include files.
-l Compile a C-preprocessed SALT file.
-b Compress runs of blank lines to one blank line.
-e Send C preprocessor error output to standard output.
-s Sets the stack size used by the SALT compiler.

EXPLANATIONS OF OPTIONS:

-Q Use CMD as preprocessor.

CSP usually uses either QCL or CL as the C preprocessor. If you have another C compiler that can produce compatible preprocessed output /Q will force CSP to use it. If the command requires switches the whole command string must be quoted. For instance if you want CSP to use CL instead of QCL (which it would do anyway if it didn't find QCL but I can't think of a better example) use:

```
CSP -Q"CL /P" FOO.SLT
```

-P Preprocess only, do not compile. Creates a .SLI file.

CSP will send the file through the C preprocessor and create a CS-compilable SLI file, but will not compile it.

-C Tell C preprocessor not to strip comments.

This passes the "/C" switch to the C preprocessor which tells it not to strip comments from the source file.

-D Tell C preprocessor to define MACRO as VALUE.

Each -DMACRO=VALUE used passes a "/DMACRO=VALUE" to the C preprocessor to define a macro.

-l Compile a C-preprocessed SALT file.

If you have another C compiler that produces Microsoft-compatible preprocessor output, you can preprocess the script first then use CSP with the -l switch on the output file. It will not preprocess the file, but will compile it with CS and change line references in CS error messages.

-b Compress runs of blank lines to one blank line.

Usually the C preprocessor replaces lines containing preprocessor directives and "switched off" code with blank lines. This option replaces multiple blank lines with one blank line, resulting in a more presentable SLI file.

-e Send C preprocessor error output to standard output.

This will force the C preprocessor to send all it's messages to the standard output, allowing it to be redirected with '>', '>>', and '|'.

-s Sets the stack size used by the SALT compiler.

This option is passed to the SALT compiler to set the stack size of the compiled SLC file.

Since you have to have a C compiler to use this I'm assuming you are either already familiar with the C preprocessor directives or have them documented elsewhere, and they won't be described here. Obviously they work here just as they do in C, since the C compiler is doing the preprocessing. Some notes:

There are a few preprocessor directives that QCL does not remove (#pragma for instance), and these are stripped by CSP before the SALT compile.

CSP will complain about code like this:

```
#define FOUR 2+2
int x=FOUR;
```

Because neither the C preprocessor nor CS replaces "2+2" with "4", so the second line actually evalutes to:

```
int x=2+2;
```

and the SALT compiler only allows variable declarations to initialize a variable to a constant, not an expression. If you can't reduce the macro to a constant then the the variable must be declared and intialized separately:

```
#define FOUR 2+2
int x;
x=FOUR;
```

Because the C preprocessor strips comments by default, SALT scripts that are preprocessed with CSP can contain C-style "/* */" comments, as long as the -C switch is not used.

DISCLAIMER:

CSP is provided on an "as is" basis without warranty of any kind, expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The person using the software bears all risk as to the quality and performance of the software. Should the software prove defective, the user assumes the entire cost of all necessary repair, servicing, or correction. The author will not be liable for any special, incidental, consequential, indirect or similar damages due to loss of data or any other reason, even if the author or an agent of the author has been advised of the possibility of such damages. In no event shall the author's liability for any damages ever exceed the price paid for the license to use the software, regardless of the form of the claim.

There is no charge for private, non-commercial use of CSP (but if you send a check anyway it won't go uncashed). If you use CSP in the development of scripts which are sold to others, a $10 registration is required. Comments, suggestions, and requests can be left at:

The Dead of Night BBS
(703) 644-7667

or mailed to:

Donald Mehrtens
5923 Minuteman Rd
Springfield VA 22152

or sent via CompuServe (which I don't check very often) to ID 72361,1407.

SALT Authors, HELP!

Is there any way for a script to determine its own name, as DOS batch files can do with %0 or C programs can do with argv[0]?

Some scripts run considerably slower if _scr_chk_key is set to 0 to disable the ESC key from prompting to abort the script. Can anything be done about this? Can the ESC key be prevented from aborting scripts without using _scr_chk_key?

Is there any way to find out whether the cursor is on or off?

Is there a cleaner way to find out if the status bar is on than to look on the screen for it with vgetchr/vgetstr?

Is there a way to see if a local key has been hit without destroying the terminal() function's ability to process it, like cinp_count() does with the comm port?

Despite the fact that Telix does not appear able to do anything with them, CS will happily compile a script that has no main() function. Is this of any use?

Can a script find out which dialing directory entries are marked? Can a script mark and unmark entries?

And does ANYBODY know ANYTHING about Telix 4.0?