(C) 1993 BASIC d.o.o Ljubljana,

Jure Spiler,

Jesenkova 5,

61000 Ljubljana, Slovenia

tel: +386 1 314 069

fax: +386 1 318 211

CompuServe: [70541,1765]

e-mail:

Jure Spiler, director jure.spiler@public1.nop[illegible]mail.si

Joze Marincek joze.marincek@uni-lj.si

Lisp2C

AutoLISP to C (ADS) treanslator user's guide

version 1.9 (22-June-1993)

1. INTRODUCTION

Lisp2Cads translates an AutoLISP source file(s) into C source files that can be further compiled using the Watcom or Metaware C compiler.

Why would anyone bother to compile the existing .LISP code?

First, this completely protects your algorithms. If you use ordinary AutoLISP, you have to provide source that can be read by AutoCAD. But then it can also be read by a human. Therefore all of your know-how is exposed to everyone interested.

Second, an ADS environment is gaining more and more acceptance. Lisp2C is a great way to preserve all your investments into Lisp (trainig, coding) and slowly moving to the ADS.

And finally, Lisp2C also includes a debugging tool that is easy to use yet powerful.

Requirements are:
AutoCAD R12 (Dos, Windows)
Watcom C/386 9.0 (9.01d required for Windows), or

Metaware C/C++ 3.1 and PharLap DOS Extender/Linker
You need DOS4GW.EXE to run L2C.EXE!

Lisp2Cads consists of:

1. QSTART.TXT	How to quick - start LISP2C
LISP2C.DOC	This file (Word for Windows)
LISP2C.TXT	This file (ASCII)
2. L2C.EXE	Lisp to C compiler
3. L2C.H	Header file, included into source
4. L2C.LIB	Libs: (Watcom -DOS)
WINL2C.LIB	(Watcom - Windows)
MWL2C.LIB	(Metaware - DOS)
MWWINL2C.LIB	(Metaware - Windows)
5. DEMO.LSP	Sample Lisp program
6. DLINE.LSP	Sample LISP from ACAD12
STARTUP.LSP	Direct statements from DLINE
DLINE.L2C	Project file to compile DLINE. LSP

Other example files may appear in distribution.

2. INSTALLATION

2.1.DOS - WATCOM

Install your Watcom C/386 9.0 or later compiler and compile at least one sample file (eg TOWER.C) from \ACAD\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place L2C.LIB and L2C.H files into \ACAD\ADS directory. Point to this directory with L2C variable:

```
    SET L2C=C:\ACAD\ADS
```

Change INCLUDE variable to include \ACAD\ADS directory:

```
    SET INCLUDE=C:\WATCOM\H;C:\
    ACAD\ADS
```

2.2.DOS - METAWARE

Install your Metaware HighC/C++ 3.1 compiler and compile at least one sample file (eg TOWER.C) from \ACAD\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place MWL2C.LIB and L2C.H files into \ACAD\ADS directory. Point to this directory with L2C variable:

```
    SET L2C=C:\ACAD\ADS
```

Change IPATH variable to include \ACAD\ADS directory:

```
SET IPATH=C:\HIGHC\H;C:\ACAD\
ADS
```

Note that paths must be set **before** Lisp file is converted into C source. L2C uses the values of these variables to produce .BAT and .MW files. If those values are not set or set properly, .BAT and .MW files may not compile the C files. This does not, however, corrupt produced C code in any way.

2.3.WINDOWS - WATCOM

Install your Watcom C/386 9.01d or later compiler and compile at least one sample file (eg TOWER.C) from \ACADWIN\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACADWIN directory.

Place L2CWIN.LIB and L2C.H files into \ACADWIN\ADS directory. Point to this directory with L2C variable:

```
SET L2C=C:\ACADWIN\ADS
```

Change INCLUDE variable to include \ACADWIN\ADS:

```
SET INCLUDE=C:\WATCOM\H;C:\
ACADWIN\ADS
```

Copy the file ADS.ICO from ADS\WIN directory to your working directory.

2.4.WINDOWS - METAWARE

Install your Metaware HighC/C++ 3.1 compiler and compile at least one sample file (eg TOWER.C) from \ACADWIN\ADS, to ensure that the compiler is set up properly.

Tip: Read README.ADS from ACADWin. Use -NOSTUB switch with PharLap linker 5.0 or later. Earlier versions of PharLap do not need this switch.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place MWL2CWIN.LIB and L2C.H files into \ACADWIN\ADS directory. Point to this directory with L2C variable:

```
    SET L2C=C:\ACADWIN\ADS
```

Change IPATH variable to include \ACADWIN\ADS directory:

```
    SET IPATH=C:\HIGHC\H;C:\
    ACADWIN\ADS
```

Note that paths must be set **before** Lisp file is converted into C source. L2C uses the values of these variables to produce .BAT and .MW files. If those values are not set or set properly, .BAT and .MW files may not compile the C files. This does not, however, corrupt produced C code in any way.

Copy the file ADS.ICO from `\ACADWIN\ADS\WIN` directory to your working directory. Again, this only affects the compilation with produced .BAT and other files.

2.5. DOS Example (Watcom)

Place the rest of the files into your working directory. These files are included only as a demonstration and can be deleted altogether.

Example:

Let us assume that you have correctly set up the Watcom C/386 9.0 to the C:\WATCOM directory. Thus, when you type SET, you might see something like

```
    PATH=...C:\ACAD;C:\WATCOM\BIN;
    C:\WATCOM\BINB;C:\WATCOM\
    LIB386\DOS;...
    WATCOM=C:\WATCOM\.
    INCLUDE=C:\WATCOM\H
```

In order to use ADS, your C:\ACAD\ADS directory should contain at least the following files:

WCADS90.LIB
ADSLIB.H
ADSDLG.H
ADS.H
ADSCODES.H

If those files are missing, you can copy them from our distribution.

Now you can copy L2C.EXE to C:\ACAD directory (listed in path), and L2C.H, L2C.LIB into C:\ACAD\ADS directory. Also, you should change the INCLUDE variable so that Watcom C/386 will search for header (.h) files also in C:\ACAD\ADS directory:

```
    SET INCLUDE=C:\WATCOM\H;C:\
    ACAD\ADS
```

And finally, you should set the L2C system variable to C:\ACAD\ADS:

```
     SET L2C=C:\ACAD\ADS
```

3. THE USAGE

3.1. COMMAND LINE INPUT

The syntax is:

```
     L2C [options] file [file...]
```

where options are

`d`	includes debugging information
`oname`	sets the output file name to 'name', instead to the name of 1st input file name
`c`	compiler (cWAT = Watcom, cMW = Metaware)
`e`	compiles every function

separately

`y`	"yes" to all questions (except for the registration)
`n`	"no" to all questions (except for the registration)
`t`	target (currently `Dos` or `WIndows`)
`?`	displays simple help

The option must be preceded with either / or - character. If an invalid option is specified, program terminates with a message.

`File` is the name of AutoLISP source file. L2C produces `file`.C file (if more than one file is specified, the first name is taken, unless /`oname` option is used). In addition, one file is produced for each (defun...) and (lambda...) These additional files have the name of the main file `file` padded with underscores ("_") to the length of total 8 (eight) characters, and then up to the last three characters are replaced with a number, starting from 0.

If two different applications are compiled on the same directory, then the corresponding main file names must differ in first five characters, or the files of one application will tackle with the files from the other application.

The order of files and options is insignificant. They can also be mixed. Example: to compile

DLINE.LSP, one could use the following command:

```
    L2C ai_utils startup /odline
    dline /d
```

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE. BAT, DLINE.MAK, and DLINE.LNK. For Windows, also DLINE.RC and DLINE.DEF would be generated.

3.2. INTERACTIVE INPUT

Alternatively, one can invoke L2C without parameters:

```
     L2C
     Input file name (.lsp):
```

The user then enters one file name per line. The input is terminated with a blank line. Then the question appears:

```
    Include debugging information
    (<Yes>/No/?):
```

The answer "Yes" is equivalent to specifying the `/d` switch. The answer "No" is equivalent to omiting that switch.

Next, the target environment and compiler are specified:

```
    Target (<Dos>/Windows/?):
    Compiler (<Watcom>/Metaware/?)
```

Currently supported are DOS and Windows 3.1 operating systems. The code produced is the same in both cases. However, the support files (. BAT, .MAK, .LNK, and optionally .RC and . DEF) files are different for those two environments. Those two questions correspond to `/t` and `/c` switches.

Next, the way how produced source code will be compiled, is choosen:

```
    Arrange for separate
    compilation of each file (Yes/
    <No>/?):
```

This question corresponds to the /e switch. The answer "Yes" is equivalent to specifying that switch, and the answer "No" is equivalent to omitting it.

Next, an answer to all questions can be specified:

```
    Answer to all questions (Yes/
    No/<Ask>/?):
```

This question corresponds to `/y` and `/n` switches. The answer "Yes" is equivalent to specifying the `/y` switch, the answer "No" is equivalent to specifying the `/n` switch, and the answer "Ask" is equivalent to omitting both two switches.

Next, the code commenting can be disabled:

```
    Comment the produced code
    (<Yes>/No/?):
```

This question corresponds to `/b` switch. The answer "Yes" is equivalent to specifying that switch and the answer "No" is equivalent to omiting it.

Finally, user can specify the output file name:

```
    Output file name (5 chrs
    significant) <>:
```

This question corresponds to an `/o` switch. In angle brackets, the name of the first file appears as a default output file name. If the default file name is longer than 5 characters, then only the first five characters are in the upper case, and the rest are in the lower case letters. One should be aware, that Lisp2Cads will use only the first 5 characters for all files but the main .C file, the .BAT file, and the .LNK file.

Example: to compile DLINE.LSP, one could use the following command:

```
L2C
Input file name (.LSP):
AI_UTILS
Input file name (.LSP): STARTUP
Input file name (.LSP): DLINE
Input file name (.LSP):
Include debugging information
(<Yes>/No/?): Yes
Target (<Dos>/Windows/?): Dos.
Compiler (<Watcom>/Metaware/?)
: Watcom
Output file name (5 char..)
<AI_UTils>: DLINE
```

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE.BAT, and DLINE.LNK.

3.3. PROJECT FILE

The third option is to invoke the compiler with the name of the project file, preceded with an @ character. Project file is an ASCII file. Each line is either a comment (starts with an * (asterisk) or ; (semicolon) in the 1st column), a file name, or an option. The syntax for options is the same as in the command line case. A sample project file (with no comments) might look like:

```
ai_utils
startup
dline
/odline
/d
/tWIN
/cWAT
```

```
/n
```

The default extension for a project file is .L2C. You can specify a full project file name, if necessary.
Example: to compile DLINE.LSP, one could use the following command:

```
L2C @DLINE
```

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE.BAT, DLINE.MAK, and DLINE.LNK.

3.4. SWITCHES

/D - Debugging

When this switch is used, a code is added to every function that prints out the function name and its arguments, and simple debugger is enabled Also, batch and link files are set to include the debugging information. You will find more information about debugger in Section 6.3.

/Oname - Output filename

You can specify the output file name. If none is specified, the first file name is taken.

/C - Compiler

Currently supported are Watcom C/386 compiler (`/CWAT`) and MetaWare High C/C++ compiler (`/CMW`). By default, compiler assumes Watcom C/386. If you use MetaWare C/C++, then use `/CMW` switch.

/E - Separate compiling

By default, all the functions are included to the main file during compile time. In this way, the C compiler has only to be loaded once, and the compilation process is significantly faster. Using `/E` switch, you force the L2C to produce several source files; one main source and one source file for every LISP function. They are compiled separately and then linked together. This way, you can edit functions without recompiling all the source code over and over again. Note that this is not simply the question of the batch and link files produced by L2C. The headers of C source files are also different.

/Y - YES to all questions

/N - NO to all questions

Specifies that answers to all the questions are Yes or No, respectively. If you specify both switches, the last is used.

These two switches do not apply to the question on registration.

/T - target system (DOS or Windows)

The target is the system under which the compiled application will be running. Currently two target systems are supported: MS-DOS and MS-Windows. To choose MS-DOS as a targeting environment, use `/TDOS` switch. To use MS-Window as a targeting system, use `/TWIN` switch. The former is default.

/B - brief code generator

Normally, Lisp2C inserts corresponding parts of Lisp code as a comment to the prodused C code. This is intended to ease the code modification process. However, the size of produced .C files is expanded significantly. If you don't intend to modify the code or you don't have enough disk space, specify `/b` switch to surpress the insertion if the comments.

/? - Help

This switch displays a simple remainder of the switches and stops the compiler execution.

3.5. COMPILING THE CODE (WATCOM)

The C source file must be translated with Watcom C/386 compiler. The object (.OBJ) file produced by Watcom C/386 compiler must be further linked together with WCADS90.LIB and L2C.LIB libraries under DOS, or with WINADS.

OBJ, WINADS.LIB and WINL2C.LIB libraries under Windows.

To simplify the job, Lisp2C translator, automatically produces several files, a batch file named `name`.BAT, make file `name`.MAK, a link file `name`.LNK, and possibly simple resource file `name`.RC and a simple definition file `name`.DEF. Batch file invokes the make utility to compile and link all files into an .EXP file under DOS, or .EXE file under Windows. The file `file`.LNK uses the system variable L2C to locate the libraries. If the user hasn't preset the variable, `file`.BAT sets it to point to a \ACAD\ADS directory (on the current drive).

If a single file is to be compiled, the command

```
     wcc386p /mf /3s /fpi87 <file>
```

can be used for a DOS target. The meanings of the options are

/mf	generate the code for the flat memory model,
/3s	pass the arguments on the stack,
/fpi87	generate in-line calls to a math coprocessor.

Similar command for Windows environment would be

```
     wcc386p /mf /3s /fpi87 /s /j /
    opmaxet /dWIN /dWATWIN /zW
    <file>
```

where

/mf	generate the code for the flat memory model,
/3s	pass the arguments on the stack,

/fpi87	generate in-line calls to a math coprocessor,
/s	remove stack overflow checks,
/j	change char default from unsigned to signed,
/opmaxet	controls several optimization parameters,
/dWIN	defines WIN symbol (as with #define),
/dWATWIN	defines WATWIN symbol, and
/zW	uses Microsoft Windows entry/exit code.

Note that, during the compilation, a compiler might issue several warning messages. They refer to undefined macro symbols (used in other systems), unreachable statements, and unreferenced variables. This is perfectly OK, as long as no error is produced.

3.6. LINKING (WATCOM)

To link the compiled files together, it is best to use the generated linker file, as all the file names must be listed. A command

```
      wlink @file.lnk
```

should do the trick. However, do not forget the "@" symbol or the file extension!

The resulting `file`.EXP file is ready to be XLOADed.

3.7. BINDING (WATCOM)

When compiling for Windows environment, the linker produces .REX file. This file has to be further binded with resource and definition files, `file`..RC and `file`.DEF, respectively. This can be achieved with the command

```
    wbind file -R file.rc file.exe
```

Make sure that the ADS.ICO file is placed in the current directory, You can find that file in \acadwin\ads directory.

Example.

To compile DEMO.LSP the command

```
    L2C DEMO
```

produces the following C source and some support files:

contains the main loop

code for (defun qsort ...)

code for (defun c:stat ...)

code for (defun c:gc ...)

code for (defun c:interpreter ...)

code for (defun S::STARTUP ...)

batch files that starts make utility,

make file,

link file used by DEMO.MAK.

is produced under Windows only,

is produces under Windows only,

Then the command

```
    DEMO
```

produces (among others) ADS module, DEMO. EXP file , which can be later loaded into AutoCAD with the command

```
    (xload "demo").
```

Under Windows, DEMO.EXE is produced, which can be XLOADed into AUTOCAD for WIndows.

4. SUPPORTED FUNCTIONS

Lisp2Cads *DEMO* supports AutoLISP R12 functions, except AME and ASE support. Here is the brief list of supported functions:

(* *error* + - / /= 1+ 1- < <= = > >= abs alloc and append apply ascii assoc atan atof atoi atom atoms-family(*) boole boundp caaaar caaadr caaar caadar caaddr caadr caar cadaar cadadr cadar caddar cadddr caddr cadr car cdaaar cdaadr cdaar cdadar cdaddr cdadr cdar cddaar cddadr cddar cdddar cddddr cdddr cddr cdr chr cond cons cos defun eq equal eval exit exp expand expt fix float foreach gc gcd getenv if itoa lambda last length list listp load log logand logior lsh mapcar max mem member min minusp not nth null numberp open or prin1 princ print progn quit quote read read-char read-line rem repeat reverse set setq sin sqrt strcase strcat strlen subst substr terpri trace type untrace ver vmon while write-char write-line zerop ~ acad_colordlg acad_helpdlg acad_strsort ads alert angle angtof angtos command cvunit distance distof entdel entget entlast entmake entmod entnext entsel entupd findfile getangle getcorner getdist getfiled getint getkword getorient getpoint getreal getstring getvar graphscr grclear grdraw grread grtext grvecs handent initget inters menucmd nentsel nentselp osnap polar prompt redraw regapp rtos setvar ssadd ssdel ssget sslength ssmemb ssname tablet tblnext tblsearch textbox textscr trans vports wcmatch xdroom xdsize xload xunload load_dialog unload_dialog new_dialog new_dialog start_dialog done_dialog term_dialog action_tile mode_tile get_attr get_tile set_tile start_list add_list end_list dimx_tile dimy_tile start_image vector_image fill_image slide_image end_image client_data_tile)

5. DIFFERENCES AND LIMITATIONS

5.1. EXPRESSIONS OUTSIDE THE FUNCTIONS

Only Lisp expressions inside Lisp functions are compiled. Other expressions are merely skipped (and a warning message is generated). They can be collected automatically into S::L2CSTARTUP function (into the file ?????__S.LSP).

5.2 NUMBER OF ARGUMENTS

User functions in Lisp2Cads can only have up to 32 arguments if they are ever to be evaluated using EVAL, APPLY or MAPCAR function.

5.3. PASSING SYMBOLS, SUBROUTINES ETC.

Currently, there is no (regular) way to exchange SYMbols, SUBRoutines, EXSUBRoutines and some other exotic data types between an ADS application and an AutoLISP environment. Functions that expect symbols as their arguments (as when calling parameters by reference) should be rewritten in a way that they would accept strings as arguments and then READ out the symbol. This only applies to a function that is called from AutoLISP. Function called directly from another L2C function can pass symbols without any limitations.

5.4. PASSING LISTS OF INTEGERS ETC.

The transfer of the objects between AutoLISP and an ADS application is not an exact one. For example, if the function `FOO` is invoked with a list of two integers as the only argument: `Command: (FOO '(1 2)),` then an ADS application will receive this as a 2D point, with integers already converted into reals. As most of the functions that expect real value they work well if given an integer argument, while the opposite might not be true, Lisp2Cads application will transform any whole element of 2D or 3D point passed from AutoLISP to integer. Therefore an unexpected results may occur every now and then.

Example:

AutoLisp value is seen by Lisp2C as