

User's Guide to the TXL Compiler / Interpreter

Version 10.6

July 2012

James R. Cordy

TXL

James R. Cordy

**User's Guide to the TXL Compiler / Interpreter
Version 10.6**

© 1991-2012 James R. Cordy

Previous versions of portions have also appeared

© 1991 Gesellschaft für Mathematik und Datenverarbeitung mbH,

© 1991-2004 Queen's University at Kingston,

© 1995-2000 Legasys Corporation, and others. Used by permission.

July 2012

Software Technology Laboratory
School of Computing
Queen's University at Kingston
Kingston, Ontario K7L 3N6
Canada

<http://www.txl.ca>

Table of Contents

1. The TXL Compiler/Interpreter	1
1.1 TXL Compiler/Interpreter Commands	1
1.2 File Arguments and Option Flags	1
2. Running TXL Programs - the 'txl' Command	3
2.1 'txl' Command Option Flags	3
2.2 'txl' Command Debugging Options	6
2.3 Accessing Command Line Arguments in TXL	7
3. Debugging TXL Programs - the 'txldb' Command	9
3.1 'txldb' Command Option Flags	9
3.2 'txldb' Command Debugging Options	10
3.3 TXL Debugger Commands	11
4. Tuning TXL Programs - the 'txlp' Command	13
4.1 'txlp' Command Option Flags	13
5. Compiling TXL Programs - the 'txlc' Command	14
5.1 'txlc' Command Option Flags	14
6. Running TXL Standalone Applications	16
6.1 Standalone Application Option Flags	16
7. Redirecting and Embedding TXL Commands and Applications	17
7.1 Redirecting TXL Commands and Applications	17
7.2 Interactive TXL Commands and Applications	19
7.2 Embedding TXL Commands and Applications	22
8. Direct Execution of TXL Program Source Files in Unix/Linux	25

1. The TXL Compiler / Interpreter

The TXL compiler/interpreter is an interpreter and compiler for TXL language programs. TXL is a programming language specifically designed to support transformational programming. The basic paradigm of TXL involves transforming input to output using a set of structural transformation rules that describe by example how different parts of the input are to be changed into output. Each TXL program defines its own context free grammar according to which the input is to be parsed, and rules are constrained to preserve grammatical structure in order to guarantee a well-formed result.

1.1 TXL Compiler/Interpreter Commands

The TXL transformation system provides four commands for compiling, executing and debugging TXL programs:

- The *txl* command invokes the TXL interpreter to compile, load and execute a TXL program on an input file.
- The *txldb* command additionally invokes the TXL rule debugger to allow stepwise execution of the TXL program in an interactive command line debugging environment.
- The *txlp* command invokes the TXL interpreter to compile, load and run a TXL program with grammar and ruleset profiling, producing a detailed time and space profile of the program's input parse and ruleset execution for a given input.
- The *txlc* command invokes the TXL compiler to translate a TXL program directly to a standalone executable application that can be run independently of TXL.

1.2 File Arguments and Option Flags

The *txl* and *txldb* commands normally take only one file argument, the name of the input file to be transformed. By convention the input file name is expected to be of the form *inputfile.dialect* where *dialect* is the name of the TXL transformation to be run on the file. The source for the TXL program for the corresponding transformation is then assumed to be in the file *dialect.Txl*. For example, if the command

```
txl Expression.Calculator
```

were to be run, the TXL processor would assume that the TXL program to be run is *Calculator.Txl*. The TXL program to be run need not be in the present working directory, but may instead be in either the user's TXL library directory, *./Txl*, or the system's TXL library directory, normally */usr/lib/txl* or *C:\windows\txl*.

For example, if the command above were run, then the program *Calculator.Txl* could be either in the present working directory (i.e., file *./Calculator.Txl*), the *Txl* subdirectory of the present working directory (i.e., file *./TxlCalculator.Txl*), or the system TXL library (i.e., file */usr/lib/txlCalculator.Txl* or *C:\windows\txl\Calculator.Txl*). In cases where an appropriate TXL source program is present in more than one of these, the user's library takes precedence over the system library, and the present working directory takes precedence over the user's library.

Files referred to by *include* statements in the TXL program are resolved in similar fashion. Included files are searched for first in the directory of the including source file, then in the *Txl* subdirectory of the present working directory, then in the system TXL library directory.

It is also possible to explicitly specify the TXL program to be run using two file arguments, for example:

```
txl count.c c2p.Txl
```

In this case the name of the input file need not be named using the suffix corresponding to the TXL program. In all cases, TXL main program files must be named using the *.Txl* suffix.

All TXL commands implement several standard options, including *-help*. The *-help* option prints a short explanation of command usage and options and halts.

2. Running TXL Programs - the 'txl' Command

```
txl [ options ] [ -o outputfile ] inputfile [ txlfile ]  
      [ -useroptions ]
```

The *txl* command provides a convenient interface for compiling, loading and executing TXL programs to transform input files. The first command argument, *inputfile*, specifies the input file to be transformed by the TXL program, and the second argument, *txlfile*, is the TXL program itself. The *txlfile* must be named ending in ".Txl" (e.g., *Transform.Txl*) and is normally either in the present working directory, the *Txl* subdirectory of the present working directory, or the TXL system library (e.g., */usr/lib/txl* or *C:\windows\txl*). If the *txlfile* is omitted, then it is inferred from the file name suffix of the given *inputfile*. For example, if the input file to be transformed is *myinput.pas* then the TXL file is inferred to be *pas.Txl*.

Unless the *-c* option is used, in which case execution is suppressed, the *txl* command compiles, loads and executes the TXL program to transform the given input file. Transformed output (only) is sent to the standard output stream (*/dev/stdout* or *CONOUT\$*) and may be saved in an output file using either the *-o* option or command line redirection (e.g., *txl inputfile txlfile > outputfile*). TXL messages are sent to the terminal independently via the standard error stream (*/dev/stderr* or *CONERR\$*), and can be redirected separately (e.g., *txl inputfile txlfile > outputfile 2> errorfile*).

2.1 'txl' Command Option Flags

The following options are recognized by the *txl* command.

-q[uiet] Quiet operation - turn off all information messages.

TXL normally prints a version identification and short progress messages indicating the stage of processing on the standard error stream as it runs. The *-quiet* option suppresses all such messages, and allows TXL to print error messages only.

-v[erbose] Verbose operation - give greater detail in information messages.

Causes TXL to print more detailed progress messages giving information on space and files used by each stage of processing. Also enables a number of common warning messages that are suppressed by default.

-c[ompile] Compile TXL program to TXL virtual machine byte code only (do not execute).

The compiled TXL virtual machine byte code for the TXL program *prog.Txl* is output to the file *prog.CTxl*. The TXL byte code file can be loaded and executed directly by subsequent *txl* commands using the *-l* option, thus avoiding the overhead of recompiling the program on every run. TXL byte code files can be converted to C and compiled to standalone TXL applications by the *txlc* command (see "[The 'txlc' Command](#)" below).

-l[oad] Load and transform input using a previously compiled TXL program.

The compiled TXL virtual machine byte code for the TXL program *prog.Txl* is loaded directly from the file *prog.CTxl*, which must be a TXL byte code file created by a previous *txl -c* command. The name of the TXL byte code file can either be inferred from the suffix of the first argument file or be explicitly given as the second argument. In either case the suffix of the TXL program file is ignored and *.CTxl* is used instead.

-d[efine] *SYMBOL* Define the TXL preprocessor symbol *SYMBOL*.

Sets the preprocessor symbol *SYMBOL* to defined, so that *#ifdef SYMBOL* preprocessor directives will succeed in the compile of the TXL program.

-comment Treat comments in the input file as input items.

Normally TXL discards any comments in the input before parsing. When this option is used, comments are treated as input items to be parsed like any other. Care must be taken to insure that the input language grammar specified in the TXL program explicitly allows comments in all the expected places, otherwise syntax errors will be flagged.

-char Treat all input characters (including newlines and spaces) as significant.

Normally TXL treats newlines, spaces and tabs ("white space") in input as separators only. When this option is specified, all characters in the input are treated as significant and categorized as tokens to be parsed according to the grammar. This option enables the predefined token classes *[space]* (any sequence of spaces and tabs) and *[newline]* (a newline character). The grammar must be crafted to accept newlines and spaces wherever they may occur in the input. This option automatically suppresses all output spacing.

-newline Treat newline characters only (not tabs and spaces) as significant.

Normally TXL treats newlines, spaces and tabs ("white space") in input as separators only. When this option is specified, newline characters (i.e. line boundaries) in the input are treated as significant and categorized as tokens to be parsed according to the grammar. This option enables the predefined token class *[newline]* (a newline character, that is, ASCII LF (Unix/Linux) or CR/LF (Windows)). The grammar must be crafted to accept newlines wherever they may occur in the input.

-multiline Allow tokens to cross line boundaries (default).

By default TXL allows tokens such as *[stringlit]* and *[charlit]* to cross line boundaries in order to handle multi-line strings, as for example in C. Newlines inside a multi-line token are preserved in the token text. For some languages and grammars it may be necessary to limit tokens to a single line by disabling this option using *-nomultiline* as a *#pragma* in the grammar.

-token Treat newlines and spaces as separators only (default).

Disables character level input, and treats all newlines, spaces and tabs as separators only. This is the default input mode.

-txl Treat the input being transformed as TXL source.

This option makes it possible to transform TXL programs themselves using TXL. Sets the lexical conventions of the input to TXL's own defaults. In particular, disables unquoted character literals and enables treatment of single quotes as separate input tokens.

-attr Print attributes in the transformed output.

By default attributes (items of type **[attr X]** for some X) are not printed in output. This option causes all attributes to be treated as regular output items.

- raw** Output transformed source in raw (unspaced) format.
- Normally TXL uses a set of built-in spacing rules appropriate to most high-level programming languages for formatting output. This option turns off all spacing and line wrapping in output except where the output would otherwise be ambiguous (e.g. between two adjacent identifiers), and as explicitly specified in the grammar using [SP] and [NL]. When used in conjunction with [SP], [NL] and [IN]/[EX], this option gives the user complete control over output formatting.
- id[chars] 'CCC'** Treat the characters 'CCC' as valid characters in [id] tokens.
- Adds the given characters as valid identifier characters in the [id] predefined token class. More general control over [id] can be achieved using the *tokens* statement in the TXL program.
- sp[chars] 'CCC'** Treat the characters 'CCC' as white space.
- Adds the given characters to the set of characters treated as equivalent to the space character and treated as separators. If *-char* is specified, these characters will be added to those matched by the *[space]* predefined token class.
- esc[char] 'C'** Use 'C' as the escape character in string and character literals.
- By default the escape character is '\ ' (e.g., *"Here: \" is an embedded quote"*). If either *"'"* or *'''* is specified, then *'''* is used for string literals (e.g., *"Here: ''' is an embedded quote"*) and *"'"* is used for character literals (e.g., *'Here: ' is an embedded quote'*).
- upper** Translate all unquoted input to upper case.
- Translates all input tokens except *[stringlit]* and *[charlit]* to upper case on input, and preserves the change in output.
- lower** Translate all unquoted input to lower case.
- Translates all input tokens except *[stringlit]* and *[charlit]* to lower case on input, and preserves the change in output.
- case** Ignore case in input.
- By default TXL is case-sensitive, that is, *abc*, *ABC* and *aBc* are all treated as different. This option specifies that the input language is case-insensitive, that is, *abc*, *ABC*, *aBc* and *Abc* are all to be treated as the same. Original case of input tokens is preserved in output unless explicitly changed by the TXL program (for example, using the *[toupper]* or *[tolower]* built-in functions). However, within the TXL program all identifiers and keywords will appear to be lower case only, so literal keywords or identifiers in nonterminal definitions, functions and rules should be coded in lower case.
- w[idth] NNN** Set the maximum output line width to NNN characters.
- By default TXL formats output in at most 80 characters per line. NNN must be a positive integer between 20 and 32767. This option has no effect when the *-raw* option is used.
- in[dent] NN** Set the output indentation increment to NN characters.
- Sets the number of character positions indented by [IN] and exdented by [EX] directives to NN characters. The default is 4 characters.

- tabn1** Output [TAB_NN] directives may force a newline.
 Allows [TAB_NN] directives to force a new line in the output if necessary to align the next output token at output column NN. This is the default.
- xml** Output as an XML parse tree.
 Outputs the result of the transformation as an XML parse tree (only). Useful in converting source text to XML syntax trees, and when debugging transformations. See also the *-Dparse* and *-Dresult* debugging options.
- i[nclude] DIR** Add DIR to the TXL include file search path.
 Adds DIR to the set of directories searched for TXL *include* files that are not present in the present working directory or its *Txl* subdirectory. The directories are searched in the order that their *-i* directives are given on the command line. If an *include* file is not found in any of these directories, then TXL system library directory (e.g., */usr/lib/txl* or *C:\windows\txl*) is searched last.
- s[ize] MMM** Set the TXL transform size to MM megabytes.
 Sets the virtual memory allocated to TXL compiler and transformer data structures to the indicated size. MM must be a positive integer between 10 and 1000 (4000 in 64-bit versions). In order to maximize transform efficiency, TXL liberally exploits the operating system's native virtual memory by artificially pre-allocating a fixed amount of static storage rather than attempting to manage storage dynamically. This strategy places static limits on the size of the input that can be processed as well as on the complexity and depth of the transformation that can be performed. This option explicitly sets the amount of static storage available and thus can be used to increase these limits. This option is normally used only when the system default limits have been exceeded or when permanently trimming the virtual memory used by a compiled TXL application. See also the *-usage* option.
- u[sage]** Report TXL resource usage statistics at the end of the run.
 Prints a table of the static limits on the various TXL internal data structures and the amount of each actually used by the TXL run. This option can be used to choose an appropriate transform size for typical input data (see the *-size* option above).
- o FILE** Write standard output to file FILE.
 Normally TXL writes the output of the run to the standard output. This option redirects output to the specified file instead.
- noOPTION** Turn command line option OPTION off.
 Explicitly turns off any command line option (e.g., *-noraw*).
- USERARGS** Pass remaining command arguments to the TXL program.
 Passes all following command line arguments to the TXL program in the predefined global variable *TXLargs*. For example, the command line :
 `txl eg.in in.Txl -s 100 - -myopt foo -otheropt`
 will initialize *TXLargs* to the *[repeat stringlit]* value `"-myopt" "foo" "-otheropt"`.

2.2 'txl' Command Debugging Options

The following TXL debugging options are recognized by *txl*. Some or all of these options may be disabled in versions of TXL that have been tuned for speed (e.g., *txlc*), but they are always available in *txldb*. All output produced by these options is sent to the terminal via the standard error stream, which can be redirected using shell redirection (e.g., *2> errorfile*).

- analyze** Invoke the TXL grammar and rule set analyzer.

Causes the TXL compiler to perform an additional set of checks on the grammar and rule set, including a check for ambiguities and potential efficiency issues in the grammar. This option may significantly slow down the compile.
- Dscan** Print the input tokens to the standard error stream in XML format.

Prints the input tokens recognized by the TXL scanner on the terminal via the standard error stream. Useful when debugging token definitions and unexpected syntax errors.
- Dparse** Print the input parse tree to the standard error stream in XML format.

Print the input parse tree on the terminal via the standard error stream. Useful when debugging grammars, or in understanding why a pattern has not matched the input.
- Dresult** Print the output parse tree to the standard error stream in XML format.

Print the output parse tree on the terminal via the standard error stream. Useful when debugging grammars, or in understanding why output is not formatted as expected.
- Dgrammar** Print the program grammar to the standard error stream as a parse tree schema in XML format.

TXL compiles input language grammars to a compact generic tree format used as both a pattern and a generator for parse trees of input. This option prints the compiled grammar tree for use in debugging subtle problems with a grammar.
- Dpattern** Print all pattern and replacement parse tree schemas to the standard error stream in XML format.

The parse tree of every pattern and replacement in the TXL program's rule set is printed on the terminal via the standard error stream. This option is useful in debugging patterns of rules that fail to match input as expected.
- Drules** Print out the names of rules to the standard error stream as they are applied.

A convenient trace of the order in which the rules and functions of the TXL program are actually invoked.
- Dapply** Print out the actual transformations made by rule applications on the terminal as they happen.

Useful for following the progress of a transformation. Transformations are output in the form $A \Rightarrow B$, where A and B are the text output form of the original and result scope of the transforming rule respectively.
- V[ersion]** Print the version of the TXL compiler/interpreter on the terminal.

Useful for checking the currently installed or accessible version of TXL.

2.3 Accessing Command Line Arguments in TXL

TXL programs are passed command line arguments as predefined TXL global variables. On every run, the following global variables are automatically predefined:

<code>TXLargs [repeat stringlit]</code>	TXL program arguments (see the “- USERARGS” option)
<code>TXLprogram [stringlit]</code>	file name of the TXL program being run
<code>TXLinput [stringlit]</code>	file name of the main input to the TXL program
<code>TXLexitcode [number]</code>	exit code for the TXL run

For example, if the command line to run the program is:

```
txl -s 20 myinput.input mytxlprogram.Txl - -myoption 2 -myotheroption
```

Then the following TXL global variables will automatically be preinitialized:

```
export TXLargs [repeat stringlit]
    "-myoption" "2" "-myotheroption"
export TXLprogram [stringlit]
    "mytxlprogram.Txl"
export TXLinput [stringlit]
    "myinput.input"
export TXLexitcode [number]
    0
```

The program can import these variables to test its command line arguments, for example:

```
import TXLargs [repeat stringlit]
deconstruct * TXLargs
    "-myoption" Value [stringlit] MoreOptions [repeat stringlit]
deconstruct * [stringlit] TXLargs
    "-myothereoption"
```

And the program can export *TXLexitcode* to explicitly set the result code of the run:

```
export TXLexitcode [number]
99
```

3. Debugging TXL Programs - the 'txldb' Command

```
txldb [ options ] [ -o outputfile ] inputfile [ txlfile ]  
[ - useroptions ]
```

The *txldb* command provides a convenient interface for compiling, loading and executing TXL programs to transform input files using the TXL debugger. The first command argument, *inputfile*, specifies the input file to be transformed by the TXL program, and the second argument, *txlfile*, is the TXL program itself. The *txlfile* must be named ending in ".Txl" (e.g., *Transform.Txl*) and is normally either in the present working directory, the *Txl* subdirectory of the present working directory, or the TXL system library (e.g., */usr/lib/txl* or *C:\windows\txl*). If the *txlfile* is omitted, then it is inferred from the file name suffix of the given *inputfile*. For example, if the input file is *myinput.pas* then the TXL file is inferred to be *pas.Txl*.

txldb compiles, loads and executes the TXL program under control of an interactive command line debugging interface to transform the given input file. Transformed output (only) is sent to the standard output stream (*/dev/stdout* or *CONOUT\$*) and may be saved in an output file using either the *-o* option or shell redirection (e.g., *txldb inputfile txlfile > outputfile*). Debugger prompts and TXL messages are sent to the terminal independently via the standard error stream (*/dev/stderr* or *CONERR\$*), and debugger commands are accepted from the terminal via the standard input stream.

3.1 'txldb' Command Option Flags

The following TXL options are recognized by *txldb*. For more detail on TXL options see the section "[The 'txl' Command](#)" above.

-q[uiet]	Quiet operation - turn off all information messages.
-v[erbose]	Verbose operation - give greater detail in information messages.
-d[efine] SYMBOL	Define the TXL preprocessor symbol SYMBOL.
-comment	Treat comments in the input file as input items.
-char	Treat all input characters (including newlines and spaces) as significant.
-newline	Treat newline characters only (not tabs and spaces) as significant.
-multiline	Allow tokens to cross line boundaries (default).
-token	Treat newlines and spaces as separators only.
-txl	Treat the input being transformed as TXL source.
-attr	Print attributes in the transformed output.
-raw	Output transformed source in raw (unspaced) format.
-id[chars] 'CCC'	Treat the characters 'CCC' as valid characters in [id] tokens.
-sp[chars] 'CCC'	Treat the characters 'CCC' as white space.
-esc[char] 'C'	Use 'C' as the escape character in string and character literals.
-upper	Translate all unquoted input to upper case.
-lower	Translate all unquoted input to lower case.
-case	Ignore case in input.
-w[idth] NNN	Set the maximum output line width to NNN characters.

<code>-in[dent] NN</code>	Set the output indentation increment to NN characters.
<code>-tabnl</code>	Output [TAB_NN] directives may force a newline.
<code>-xml</code>	Output as an XML parse tree.
<code>-i[nclude] DIR</code>	Add DIR to the TXL include file search path.
<code>-s[ize] MM</code>	Set the TXL transform size to MM megabytes.
<code>-u[sage]</code>	Report TXL resource usage statistics at the end of the run.
<code>-o FILE</code>	Write standard output to file FILE.
<code>-noOPTION</code>	Turn command line option OPTION off.
<code>- USERARGS</code>	Pass remaining command arguments to the TXL program.

3.2 ‘txldb’ Command Debugging Options

The following TXL debugging options are recognized by *txldb*. For more details see “[‘txl’ Command Debugging Options](#)” above. All output produced by these options is sent to the terminal via the standard error stream.

<code>-analyze</code>	Invoke the TXL grammar and rule set analyzer.
<code>-Dscan</code>	Print the input tokens to the standard error stream in XML format.
<code>-Dparse</code>	Print the input parse tree to the standard error stream in XML format.
<code>-Dresult</code>	Print the output parse tree to the standard error stream in XML format.
<code>-Dgrammar</code>	Print the program grammar to the standard error stream as a parse tree schema in XML format.
<code>-Dpattern</code>	Print all pattern and replacement parse tree schemas to the standard error stream in XML format.
<code>-Drules</code>	Print out the names of rules to the standard error stream as they are applied.
<code>-Dapply</code>	Print out the actual transformations made by rule applications on the terminal as they happen.
<code>-V[ersion]</code>	Print the version of the TXL compiler/interpreter on the terminal.

3.3 TXL Debugger Commands

The *txldb* command provides an interactive interface for stepping through a transformation on a rule-by-rule basis. Once the TXL program is compiled and loaded and execution begins, the TXL rule debugger is automatically entered and continued execution of the TXL program proceeds under control of a small set of interactive debugging commands.

The TXL debugger provides the following commands:

rules

List the names of all of the rules and functions in the TXL program on the terminal.

rule

Print the name of currently executing rule or function on the terminal.

set [RuleName]

Set a breakpoint at rule or function *RuleName* (default current rule). TXL will return control to the debugger whenever a breakpoint is encountered.

clear [RuleName]

Clear the breakpoint at rule or function *RuleName* (default current).

clr [RuleName]

Same as *clear*.

showbps

Print a list of all currently set breakpoints on the terminal.

scope

Print the text of the scope of application of the current rule invocation on the terminal. Valid only on entry to a rule, before a pattern match has been found.

match

Print the text of the current pattern match on the terminal. Valid only after pattern match and before replacement in a rule.

matchcontext

Print the text of the current pattern match highlighted in the context of the scope on the terminal. Valid only after pattern match and before replacement in a rule. The pattern match is bracketed in the scope using the markers `|>>>|` and `|<<<|`.

result

Print the text of the result of the current construct or rule replacement.

vars

Print a list of the names and types of all currently visible TXL variables.

VarName or 'VarName

Print the text of the current binding of TXL variable *VarName* on the terminal. The variable name must be quoted only if it is the same as a debugger command.

tree VarName or tree 'VarName

Print the parse tree of the current binding of TXL variable *VarName* on the terminal in XML format. The variable name must be quoted only if it is one of *scope*, *match* or *result*.

tree scope
 Print the parse tree of the scope of application of the current rule invocation on the terminal in XML format. Valid only on entry to a rule, before a pattern match has been found.

tree match
 Print the parse tree of the current pattern match on the terminal in XML format. Valid only after a pattern match and before replacement in a rule.

tree result
 Print the parse tree of the result of the current construct or rule replacement on the terminal in XML format.

where
 Print the current rule name and execution state on the terminal.

show [RuleOrDefName]
 Print the source code of the rule, function or nonterminal type definition *RuleOrDefName* (default current rule) on the terminal.

go or run
 Continue execution until the next breakpoint or end of transformation.

next or .
 Continue execution until the next statement (*construct, deconstruct, import, export, where* or *by* clause) in the current rule or function.

/RuleName
 Continue execution until the next main pattern match of *RuleName* (default current rule), or end of transformation.

/
 Continue execution until the next main pattern match of the current rule, or end of transformation.

//
 Continue execution until next pattern match (of any rule), or end of transformation.

step N
 Step trace execution for *N* (default 1) steps.

step
 Step trace execution for one step.

RETURN
 Same as *step*.

help
 Print a summary of TXL debugger commands on the terminal.

quit
 Abort the transformation and exit TXL.

4. Tuning TXL Programs - the 'txlp' Command

```
txlp [ profoptions ] [ txloptions ] inputfile [ txlfile ] [ - useroptions ]  
txlp profoptions
```

The *txlp* command provides a convenient interface for profiling the grammar and transformation rules of a TXL program for a given input. *txlp* compiles, loads and executes the the TXL program using a special profiling version of the TXL interpreter, and outputs the result profile to the terminal on the standard error stream. By default, a profile of the rule set is output, sorted by cumulative time spent executing each rule and its subrules. Using the *-parse* command line option, a profile of the grammar in parsing the input is output instead. Options allow sorting of the profile output by time, space, number of search cycles, or parse efficiency.

When run with profiling options only (e.g., *txlp -space*), *txlp* re-interprets the results of the previous *txlp* run as indicated by the new profiling options. This is done by re-reading the raw profile data stored in the files *txl.pprofout* and *txl.rprofout*, which are created by *txlp* when it is first run.

4.1 'txlp' Command Option Flags

The following profiling options are recognized by *txlp*.

-parse	Output a grammar parsing profile rather than the default rule transformation profile.
-time	Sort the profile by most cumulative time per rule or nonterminal (the default).
-space	Sort the profile by most cumulative space used per rule or nonterminal.
-calls	Sort the profile by most invocations per rule or nonterminal.
-cycles	Sort the profile by most search/ match cycles per rule or most parse cycles per nonterminal.
-eff	Sort the profile by least parse efficiency per nonterminal.
-percall	Show average time, space and cycles per rule invocation or nonterminal call.

5. Compiling TXL Programs - the 'txlc' Command

```
txlc [ txloptions ] program.Txl
```

The *txlc* command provides a convenient interface for compiling a TXL program to a standalone executable which can be distributed and executed independently of TXL. The single command argument *program.Txl* is the TXL program to be compiled. The TXL program file must be named ending in the suffix ".Txl" and must be either in the current working directory, in the *Txl* subdirectory of the current directory, or in the system TXL library (e.g., */usr/local/lib/txl* or *C:\windows\txl*).

txlc uses the TXL interpreter (see "[The 'txl' Command](#)" above) to compile the program to a TXL virtual machine byte code file, and then runs the TXL byte code converter to produce a C program that is compiled and linked with the TXL virtual machine to produce the output file *program.x* or *program.exe*, a standalone executable program. *txlc* requires that the command line C compiler and linker (normally *cc* or *gcc* on Unix and Linux systems, *xlc* on AIX, and the *MS Visual C++ 6.0* command line tools on Windows) are installed and available in the current command line environment.

All TXL information and error messages from the compile are sent to the terminal via the standard error stream (*/dev/stderr* or *CONERR\$*).

5.1 'txlc' Command Option Flags

The following TXL options are recognized by *txlc*. See "[The 'txl' Command](#)" above for more detailed explanations. TXL options specified to *txlc* become the defaults of the compiled result standalone application. Most can be overridden each time that the application is run by specifying new options on the application's own command line (see "[Standalone Application Options](#)" below).

- | | |
|-------------------------|--|
| -q[uiet] | Quiet operation - turn off all progress messages when the application is run.
This is the default for standalone applications. |
| -v[erbose] | Verbose operation - give detailed TXL information messages when the application is run. |
| -d[efine] SYMBOL | Define the TXL preprocessor symbol SYMBOL. |
| -comment | The application is to treat comments as input items.
By default standalone applications discard comments when parsing their input. |
| -char | The application is to treat all input characters (including newlines and spaces) as significant. By default applications treat newlines and spaces as separators only. |
| -newline | The application is to treat newlines only (not tabs and spaces) as significant.
By default applications treat newlines and spaces as separators only. |
| -multiline | The application should allow tokens to cross line boundaries.
This is the default for standalone applications. |
| -token | The application is to treat newlines and spaces as separators only.
This is the default for standalone applications. |

<code>-txl</code>	The application is to treat its input as TXL source. This option makes it possible to make applications that transform TXL programs.
<code>-attr</code>	The application is to include attributes in its transformed output. By default, attributes (items of type [attr X] for some X) are not printed in the output of a standalone application.
<code>-raw</code>	The application is to output transformed source in raw (unspaced) format.
<code>-id[chars]</code>	<code>'CCC'</code> The application is to treat the characters 'CCC' as valid characters in [id] tokens.
<code>-sp[chars]</code>	<code>'CCC'</code> The application is to treat the characters 'CCC' as white space.
<code>-esc[char]</code>	<code>'C'</code> The application should use 'C' as the escape character in string and character literals.
<code>-upper</code>	The application should translate all unquoted input to upper case.
<code>-lower</code>	The application should translate all unquoted input to lower case.
<code>-case</code>	The application should ignore case in input.
<code>-w[idth]</code>	NNN Set the default maximum output line width of the application to NNN characters.
<code>-in[dent]</code>	NN Set the output indentation increment of the application to NN characters (default 4).
<code>-tabnl</code>	[TAB_NN] directives may force a newline in the application's output (default).
<code>-i[nclude]</code>	DIR Add DIR to the TXL include file search path.
<code>-s[ize]</code>	MM Set the application's default TXL transform size to MM megabytes.
<code>-u[sage]</code>	Report TXL resource usage statistics at the end of every run of the application.
<code>-noOPTION</code>	Turn command line option OPTION off.

6. Running TXL Standalone Applications

```
program.x [ txloptions ] [ useroptions ] [ -o outputfile ] inputfile  
program.exe [ txloptions ] [ useroptions ] [ -o outputfile ] inputfile
```

TXL applications compiled using *txlc* are run as shown above. Transformed output is sent to the standard output (*/dev/stdout* or *CONOUT\$*) and may be redirected to a file using the *-o* option or shell redirection (e.g., *program.x inputfile > outputfile*). By default all TXL information messages are turned off and only error messages are sent to the terminal via the standard error stream (*/dev/stderr* or *CONERR\$*). If the *-v* option is used, then TXL information messages are also sent to the terminal via the standard error stream.

6.1 Standalone Application Option Flags

The following TXL options are recognized by applications created using *txlc*. In all cases, options specified on the application command line override any defaults set when the application was compiled. See "[The 'txl' Command](#)" above for detailed explanations of the options.

```
-h[elp]  
-q[uiet]  
-v[erbose]  
-w[idth] NNN  
-s[ize] MM  
-o FILE  
-V[ersion]
```

Command line options other than those above are assumed to be user program options and are passed to the standalone program in the TXL global variable *TXLargs* (see "[Accessing Command Line Arguments in TXL](#)" above).

7. Redirecting and Embedding TXL Commands and Applications

TXL commands and applications are designed to be easily integrated into interactive interfaces, embedded environments, scripting languages and compiled programs using synchronous interaction via the standard input and standard error streams. For this reason the standard input and standard error streams of *txl*, *txldb* and compiled TXL applications are unbuffered and fully synchronous. So for example, embedding the TXL debugger *txldb* in an interactive IDE environment is relatively easy.

7.1 Redirecting TXL Commands and Applications

Input and output of TXL commands and applications can be easily redirected using command line redirection or command shell stream pseudo-file names. In the following examples, the *txl* command is used, but the same techniques apply to *txldb* and compiled applications.

Saving TXL output to a file. The main output of a TXL transformation is always written to the standard output stream (*/dev/stdout* or *CONOUT\$*) and can be redirected to files using standard command shell redirection operators. For example:

```
txl inputfile txlfile.Txl > outputfile
```

The TXL *-o* command line option has the same effect.

```
txl -o outputfile inputfile txlfile.Txl
```

Piping TXL output to another command. Output can be piped to other commands using standard command shell pipes. For example, if we are only interested in certain parts of the output of a transformation, we can pipe the output to *grep* to select those.

```
txl inputfile txlfile.Txl | grep "interesting patterns"
```

The standard output stream of TXL commands and applications is designed for large scale transformed output only and therefore is fully buffered, making it inappropriate for interaction. If interactive input/output is required, the program must be modified to use programmed output instead, using the built-in input/output functions provided for the purpose (*[print]*, *[put]*, *[putp]*, *[message]* and so on).

Redirecting TXL messages and errors. TXL writes all information messages, error messages, *txldb* command prompts and so on to the standard error stream (*/dev/stderr* or *CONERR\$*). On most systems, these messages can be saved in a file using shell redirection of the standard error stream (stream 2).

```
txl inputfile txlfile.Txl > outputfile 2> errorfile
```

Some systems allow piping of the standard error stream as well, using *"2 |"*.

Programmed output in TXL programs (using *[print]*, *[put]*, *[putp]*, *[message]* and so on) is also output to the standard error stream and can be similarly separated and redirected.

Redirecting combined TXL output with messages and errors. Sometimes it's desirable to be able to save all TXL output, including messages, transformation output and programmed output, together as they would appear on the terminal. On Windows and on Unix and Linux systems using the C shell (*csh*), this can be easily done using stream cloning:

```
txl inputfile txlfile.Txl > alloutputfile 2>&1
```

Other shells (not *cs*h or Windows) can sometimes be fooled into putting all the output together using subshell bracketing:

```
(txl inputfile txlfile.Txl) > alloutputfile
```

Piping combined TXL output with messages and errors. Although it's not clear why one might want to do this, it is possible on most systems to pipe the entire TXL output and messages to another program, by redirecting the standard error stream to the standard output in a subshell and then piping the entire result:

```
(txl inputfile txlfile.Txl 2>&1) | grep "interesting patterns"
```

Transforming piped standard input. TXL commands and applications can be used in multi-stage pipelines by transforming the standard input. On Unix and Linux systems this is easy, using */dev/stdin* as the input file:

```
othercommand | txl /dev/stdin txlfile.Txl
```

Unfortunately Windows has no */dev/stdin* concept in its command shell, *CONIN\$* refers directly to the shell console and not to the standard input. For this reason, TXL commands and applications interpret the special input file name *stdin* to refer to the standard input. Thus on Windows (and for consistency on any other system as well) the above can be written as:

```
othercommand | txl stdin txlfile.Txl
```

Piped standard input makes chains of TXL transformations and mixed tool transformations easy:

```
txl originalinput stage1.Txl | txl /dev/stdin stage2.Txl |  
    txl /dev/stdin stage3.Txl > finaloutput
```

```
txl originalinput transform1.Txl | grep "interesting patterns" |  
    txl /dev/stdin transform2.Txl > finaloutput
```

```
grep "interesting patterns" originalinput |  
    txl /dev/stdin transform.Txl > finaloutput
```

7.2 Interactive TXL Commands and Applications

TXL is designed to allow for creation of interactive TXL commands and applications. In this sections we demonstrate a few simple paradigms for interaction with TXL programs. Many others can be implemented using more complex programs.

Interactive TXL programs. TXL's programmed input/output built-in functions *[put]*, *[putp]*, *[message]*, *[print]*, *[get]* and *[getp]* provide unbuffered interactive behaviour through the the standard input (*/dev/stdin*) and standard error (*/dev/stderr*) streams, which can be used to make interactive TXL programs.

For example, the following main program reads in lines of input consisting of sequences of numbers and interactively outputs the sum of each line. The program halts when an empty sequence is input (implemented by a deconstructor guard).

```
% This program does not expect any main input to transform
define program
  [empty]
end define

% This rule intentionally goes on forever,
% repeatedly replacing its empty main input by itself -
% the real work is in the interactive part
rule main
  replace [program]
    E [empty]
    % The interactive part - at each cycle, we ask for new
    % interactive input
    construct Input [repeat number]
      [getp "Numbers to add: "]
    % If it's an empty input, we're done and halt
    deconstruct not Input
      % empty
    % Otherwise add the input numbers, output the result and continue
    construct Output [number]
      _ [+ each Input] [print]
    by
      E
end rule
```

When this program is run, it interacts with the user on each cycle. The *-q* command flag tells TXL not to output its usual information messages, and */dev/null* is the Unix / Linux empty stream:

```
txl -q /dev/null sum.Txl
Numbers to add: 10 20 30 40
100
Numbers to add: 1.1 2.34 3.14159 77.3
83.8816
Numbers to add: 1024 2048 512 128 2
3714
Numbers to add:
```

This paradigm can be generalized to any transformation whose interactive inputs can be given on one line. For larger scale interaction, it is necessary to use input files, specified interactively or created by the controlling program or script.

Interactive processing of input files. The program above can be generalized to handle entire files of numbers on any number of input lines, as shown below in a program to average all the numbers in a file:

```
% This rule intentionally goes on forever,
% repeatedly replacing its empty main input by itself
rule main
  replace [program]
    E [empty]
    % The interactive part - at each cycle, we ask for a new file
    construct InputFile [stringlit]
      _ [getp "File name: "]
    % If the file name given is "quit", we're done
    deconstruct not InputFile
      "quit"
    % Read the input from that file
    construct Input [repeat number]
      _ [read InputFile]
    % Average the numbers, output the result and continue
    construct Ninput [number]
      _ [length Input]
    construct Output [number]
      _ [+ each Input] [/ Ninput] [print]
  by
    E
end rule
```

When run, the program expects a file name for each interactive input, and averages the numbers in the file to make the interactive output:

```
txl -q /dev/null avgfile.Txl
File name: "458marks"
75.4545
File name: "327marks"
71.1765
File name: "quit"
```

Run time selection of transformation. Interaction can also be used to select among a number of transformations, with the selection begin specified either through the main input, or interactively. In this simple example we interactively ask what transformation is needed, summing or averaging. The technique generalizes to arbitrarily complex sets of transformations implemented in a single interactive TXL program.

```
% This rule intentionally goes on forever,
% repeatedly replacing its empty main input by itself
rule main
  replace [program]
    E [empty]
    % The interactive part - at each cycle, we ask for new
    % interactive input, and which operation to do
    construct Input [repeat number]
      _ [getp "Numbers: "]
    % If it's an empty input, we're done
    deconstruct not Input
      % empty
    % Otherwise ask what operation is needed
    construct Operation [id]
      _ [getp "Operation (sum/avg): "]
```

```

        % Do the requested operation, output the result and continue
        construct Output [number]
            _ [sum Operation Numbers] [avg Operation Numbers] [print]
        by
            E
    end rule

    function sum Operation [id] Numbers [repeat number]
        % If the sum operation is requested
        deconstruct Operation
            'sum
        % Then add up the numbers
        construct Sum [number]
            _ [+ each Numbers]
        % And return the sum as result
        replace [number]
            _ [number]
        by
            Sum
    end function

    function avg Operation [id] Numbers [repeat number]
        % If the average operation is requested
        deconstruct Operation
            'avg
        % Then average the numbers
        construct Ninput [number]
            _ [length Input]
        construct Average [number]
            _ [+ each Input] [/ Ninput] [print]
        % And return the average as result
        replace [number]
            _ [number]
        by
            Average
    end function

```

This program interacts in two different ways when run, one to specify the data to be transformed (which could as easily be by filename as in the previous example), and one to specify the transformation. The transformation functions, in this case very simple, can be arbitrarily complex, making the paradigm completely general.

```

txl -q /dev/null sumavg.Tx1
Numbers: 2 4 6 8
Operation (sum/avg): sum
20
Numbers: 25 45 95 101
Operation (sum/avg): avg
66.5
Numbers: 2.4 6 99 73 45 90 21 4.67
Operation (sum/avg): avg
42.6337
Numbers:

```

TXL transformation servers can be implemented using this paradigm by creating an interactive TXL program that takes file names for both input and output, and operation identifiers to specify the desired transformations. The program can then be attached to an input stream such as a Unix / Linux named pipe to accept and service transformation requests on an online basis.

7.3 Embedding TXL Commands and Applications

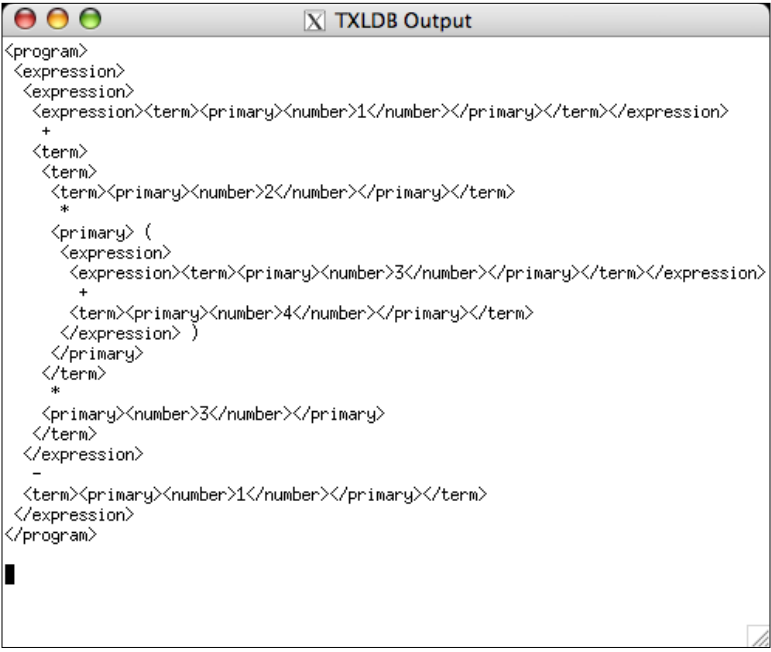
Using the paradigms above, and by attaching the standard input, output and error streams of TXL commands or applications to programmed streams in scripting or compiled languages such as *Bash*, *Csh*, *Perl*, *PHP*, *VBScript*, *Javascript*, *Tcl*, *Java*, *Visual Basic*, *C*, *C++* or so on, it is possible to create windowed interactive applications, interfaces and debugging environments embedding TXL, TXLDB and compiled TXL applications.

Windowed interactive TXLDB. As an example, it is easy to create a simple windowed interface to TXL by attaching its input and output streams to named pipes using the *Csh* scripting language, and using interprocess communication to attach these scripts to *X-windows* in a Unix / Linux environment. The example demonstrates embedding TXL in a windowed interactive application environment such as an IDE. It is intended as a demonstration only, not a real application. In this case TXLDB is embedded in a windowed environment with a separate filtered window for each of interactive command input and interactive TXLDB output.

Although this application embeds TXLDB, any TXL program using interactive input/output (i.e., the *[put]*, *[get]* and other interactive input/output built-in functions) can be similarly embedded in a windowed environment using the same techniques. The application is invoked using the *txldb.csh* command in place of *txldb* from any *Xterm* shell window, for example:

```
txldb.csh Ultimate.Question
```

The application creates two windows, one for the filtered TXLDB interactive output, and one for the filtered TXLDB interactive input commands. Commands are entered in the input command window and the results appear in the output window. The command input window is cleared when ready for a new command, giving a clear line in which to enter the new command. The output window is cleared before each reaction from TXLDB, focussing on the result of the most recent command, as shown below for the result of the “tree scope” command. The example is easily modified to be more intelligent in its processing and windowing of the interaction, or to embed an interactive TXL program in place of TXLDB.



```
<program>
<expression>
  <expression>
    <expression><term><primary><number>1</number></primary></term></expression>
    +
    <term>
      <term>
        <term><primary><number>2</number></primary></term>
        *
        <primary> (
          <expression>
            <expression><term><primary><number>3</number></primary></term></expression>
            +
            <term><primary><number>4</number></primary></term>
          </expression> )
        </primary>
      </term>
    </term>
    *
    <primary><number>3</number></primary>
  </term>
</expression>
-
  <term><primary><number>1</number></primary></term>
</expression>
</program>
```



The application and filter processes are all implemented as Csh ("C shell") scripts, but could just as easily be coded in C, C++, Java, Visual Basic, Perl, Javascript, VB script, Applescript or any other language. The windows are implemented as Xterm windows in X-windows, but could just as easily use any other windowing system on any platform, including windows or frames in a web browser.

The main script, *txldb.csh*, creates the communications channels as Unix / Linux named stream pipes, forks and attaches the pipes to separate windows for input and output, and then invokes *txldb* with its input and output redirected to the pipes. This creates the windows shown above and initializes *txldb* and the filter scripts for each window.

```
#!/bin/csh
# Trivial windowed TXLDB interface using C shell scripts and Xterm
# Usage: txldb.csh [txloptions] inputfile [txlfile]

# Make the input and output stream pipes for TXLDB
mkfifo txldbbin$$ txldbout$$

# Fork a separate Xterm window process displaying filtered interactive
# output and attach it to the TXLDB output stream pipe
xterm -title "TXLDB Output" -geometry 82x30-0+40 \
    -e "txldbout.csh < txldbout$$" &

# Fork a separate Xterm window process accepting filtered interactive
# commands and attach it to the TXLDB input stream pipe
xterm -title "TXLDB Command" -geometry 82x1-0+460 \
    -e "txldbbin.csh > txldbbin$$" &

# Run TXLDB with interactive input/output redirected to those processes
txldb $* < txldbbin$$ >& txldbout$$

# Clean up when TXLDB exits
/bin/rm -f txldbbin$$ txldbout$$
kill %1 %2
```

The output filter script *txldbout.csh* looks for output lines beginning with the TXLDB prompt, which indicate the beginning of a new interactive output from TXLDB. At each such prompt it strips out the prompt itself and clears the screen before displaying the lines of interactive output.

```
#!/bin/csh
# Simple TXLDB interactive output filter
# Display the output from each TXLDB command in a clear window

while (1)
    # Get a line of output from TXLDB
    set output="$<"

    # If it starts with a command prompt, clear the window
    # and strip the prompt
    if (`expr "$output" : "\ (TXLDB\) .*" ` == "TXLDB") then
        clear
        set output="`expr '$output' : 'TXLDB >> \(.*\) '`"
    endif
```

```

    # In any case, display it
    echo "$output"
end

```

The input filter script looks for output lines beginning with the TXLDB prompt, which indicate the beginning of a new interactive output from TXLDB. At each such prompt it deletes the prompt itself and clears the screen before displaying the lines of interactive output.

```

#!/bin/csh
# Simple TXLDB interactive command input filter
# Accept one command at a time, clear the window in between

while (1)
    # Get one command
    set input="$<"

    # Clear this window for the next one
    clear > /dev/stderr
    # Pass the command on to TXLDB
    echo "$input"
end

```

This simple example only serves to demonstrate one technique for embedding TXL commands and applications. The filtering done by the input and output interaction scripts can be arbitrarily complex, and in particular can themselves automatically initiate other interaction with the TXL command or application, for example to automatically update another window displaying bound TXL variables for TXLDB. If a TXL program is designed to interact with such an embedded environment, communications can involve more complex messaging.

8. Direct Execution of TXL Source Files in Unix/Linux

```
program.Txl [ txloptions ] [ useroptions ] [ -o outputfile ] inputfile
```

On Unix-compatible systems such as Linux, MacOS X and Solaris, TXL programs can be made directly executable using a `#!` header line in the TXL source file. For example, the following TXL program *square.Txl* can be directly executed in these systems:

```
#!/usr/local/bin/txl -q
# square.Txl: Example directly executable TXL source program
# Usage: square.Txl inputfile

define program
    [number]
end define

function main
    replace [program]
        N [number]
    by
        N [* N]
end function
```

The header specifies that the file is to be executed by the TXL interpreter */usr/local/bin/txl*, which must be given as an absolute command path (i.e., one beginning with `/` as above). Command line options that are always to be passed to the interpreter when the program is run are also specified in the header (e.g., the `-q` flag above).

In order to use direct execution, the TXL source file must have the executable (x) flag set in its file mode, for example using the command:

```
chmod +x square.Txl
```

When a directly executable TXL file is run, for example using the command:

```
square.Txl eg.square
```

The Unix -compatible system will invoke the command given in the `#!` header, with the TXL source file, any arguments and flags given in the header, and any arguments and flags given on the command line. For example, for the command above, the system will actually run the command:

```
/usr/local/bin/txl square.Txl -q eg.square
```

Directly executable TXL programs can be used as filters by specifying *stdin* as the input file in the header, for example if we change the header in *square.Txl* to:

```
#!/usr/local/bin/txl -q stdin
```

Then we can invoke it as a filter, for example as in:

```
echo 5 | square.Txl
```

An advantage of directly executing TXL programs in this way is that flags that must normally be given on the command line or permanently set in compiled standalone applications, such as *-size*, can be given in the execution header of the TXL source file itself, for example:

```
#!/usr/local/bin/txl -size 100
```

Directly executable TXL source programs have the advantage that they are platform-independent across Unix-compatible platforms, whereas compiled standalone TXL programs are platform-dependent.