Coding Conventions for TXL

TXL Working Paper 6 (Also published as Legasys WP97-103-JRC)
James R. Cordy
June 1997

Until recently, with notable exceptions, most TXL programs were authored and maintained by single programmers. Thus style, coding conventions and readability of TXL programs was not, for the most part, a major issue. However, recently more and more TXL programs, having made it to production, are rapidly passed around between several different people involved in bug fixes (usually developers), feature enhancements (usually researchers), and integration issues such as base grammar extensions (developers or researchers). Thus it is important that the style and format of the programs be quickly recognizable to many people.

Over the years a standard TXL style has evolved by example, and most TXL programmers now use that style in their TXL coding without thinking about it. However, exceptions to the style frequently crop up and can cause confusion and difficulties in understanding and maintenance, and new programmers may not notice that the style exists at all. The purpose of this paper is to document this style and its coding conventions in the hope that we can keep TXL programs easily passed from person to person and thus more easily maintainable in the long term.

File Naming Conventions

The evolved de-facto standard for naming of TXL files is as follows:

Main Programs	`Example.Txl`
Grammars	`Example.Grammar`
Grammar Overrides	`ExampleOverrides.Grammar`
Rule Includes	`Example.Rul`

Naming Conventions

The evolved de-facto standard for naming of entities in TXL programs is as follows:

Nonterminals: `[lower_case_with_underscores]`

(Exception: nonterminals implementing keyword choices may use upper case for the keywords, e.g., `[IS_or_ARE]`)

Rules: `[lowerUpperCaseBeginningWithLower]`

No underscores.

Variables: `UpperLowerCaseBeginningWithUpper`

No underscores.

Format Conventions

The evolved de-facto standard for formatting of entities in TXL programs is as follows:

Nonterminals:

```
define example_nonterminal
        [indented_eight_spaces]
    |   [alternative_bar_indented_four_spaces]
end define
```

Rules:

```
rule exampleRule
    % Comment concerning intention of rule
    import Global [indented_four_spaces_at_beginning_of_rule]

    replace [indented_four_spaces]
        Pattern [on_new_line_indented_eight_spaces]

    deconstruct * Pattern [indented_four_spaces]
        SubPattern [on_new_line_indented_eight_spaces]

    where
        Variable [onNewLineIndentedEightSpaces]

    construct Result [indented_four_spaces]
        ConstructedItems [onNewLineIndentedEightSpaces]
        Formatted [in]
            Standard [target_language]
        Indentation

    by
        Result [indentedEightSpaces]
               [formattedInStandardTargetLanguageIndentation]
               [multipleRuleCallsArrangedVertically]
end rule
```

Program Structure and Headers

The evolved de-facto standard for source structure and headers in TXL programs is as follows:

```
% Example Meaningful Transformation
% I.M. Author, DD MMM 19YY

% Copyright 19xx I.M. Author

% Revision history:

% 1.2 Revised meaningfully again  - TRD  DD MMM 19YY
% 1.1 Revised meaningfully        - JRC  DD MMM 19YY
% 1.0 Initial revision            - IMA  DD MMM 19YY


% This program does really neat stuff.
% It begins with this meaningful and comprehensive comment
% concerning its intention and the method by which it attempts
% to achieve its intention.

% The input to this program consists of lots of interesting bits.

% The output of this program is the answer to the ultimate
% question, formatted in one column.

% The following options are available:
%    -better  Achieve our intention even better than usually.


% Base grammar, and grammar override files

include "Cpp.Grammar"
include "CppOverrides.Grammar"


% Local grammar overrides

define working_paper
      ...
   |  [txl_coding_standards_working_paper]
end define


% Auxiliary rule sets

include "Oracle.Rul"
```

```
% Main rule, followed by other rules in topological order

function main
    % Begin by announcing the program
    construct Announcement [id]
        _ [message "MeaningfulTransformation (DD MMM 19YY)"]

    % Definition of all global variables used in the transform
    % should be in the main rule, even if they start empty
    export ConstantList [repeat number]
        1  2  4  8  16  32
    export ProcessedSymbols [repeat id]
        % Begins empty

    replace [program]
        AllInput [program]

    % Globals derived from the matched input go here
    export AllInput

    by
        AllInput
            % Comment describing processing step 1
            [step1Rule1]
            [step1Rule2]

            % Comment describing processing step 2
            [step2Rule1]
            [step2Rule2]
end function


% Step 1. Comments describing the workings of processing step 1

rule step1Rule1
  . . .
end rule

. . .


% Step 2. Comments describing the workings of processing step 2

rule step2Rule1
  . . .
end rule

. . .


% Utility rules supporting the above

. . .
```