Tips for writing a good JavaCC lexical specification

There are many ways to write the lexical specification for a grammar. But the performance of the generated token manager varies significantly depending on how you do this. Here are a few tips:

- Try to specify as many String literals as possible. These are recognized by a Deterministic Finite Automata (DFA), which is much faster than the Nondeterministic Finite Automata (NFA) needed to recognize other kinds of complex regular expressions. For example, to skip blanks/tabs/newlines,

  ```
  SKIP : { " " | "\t" | "\n" }
  ```

  is more efficient than doing

  ```
  SKIP : { < ([" ", "\t", "\n"])+ > }
  ```

  because in the first case you only have string literals, it will generate a DFA whereas for the second case it will generate an NFA.

- Try to use the pattern ~[] just by itself as much as possible. For example, doing a

  ```
  MORE : { < ~[] > }
  ```

  is better than doing

  ```
  TOKEN : { < (~[])+ > }
  ```

  of course, if your grammar dictates that one of these cannot be used, then you don't have a choice, but try to use < ~[] > as much as possible.

- Specify all the String literals in the order of increasing length, i.e., all shorter string literals before longer ones. This will help optimizing the bit vectors needed for string literals.

- Try to minimize the use of lexical states. When using these, try to move all your complex regular expressions into a single lexical state, leaving others to just recognize simple string literals.

- Try to use IGNORE_CASE judiciously. Best thing to do is to set this option at the grammar level. If that is not possible, then try to have it set for *all* regular expressions in a lexical state. There is heavy performance penalty for setting IGNORE_CASE for some regular expressions and not for others in the same lexical state.

- Try to SKIP as much possible, if you don't care about certain patterns. Here, you have to be a bit careful about EOF. seeing an EOF after SKIP is fine whereas, seeing an EOF after a MORE is a lexical error.

- Try to avoid specifying lexical actions with MORE specifications. Generally every MORE should end up in a TOKEN (or SPECIAL_TOKEN) finally so you can do the action there at the TOKEN level, if it is possible.

- Also try to avoid lexical actions and lexical state changes with SKIP specifications (especially for single character SKIP's like " ", "\t", "\n" etc.). For such cases, a simple loop is generated to eat up the SKIP'ed single characters. So obviously, if there is a lexical action or state change associated with this, it is not possible to it this way.

- Try to avoid having a choice of String literals for the same token, e.g.

```
< NONE : "\"none\"" | "\'none\'" >
```

Instead, have two different token kinds for this and use a nonterminal which is a choice between those choices. The above example can be written as :

```
  < NONE1 : "\"none\"" >
|
  < NONE2 : "\'none'\" >
```

and define a nonterminal called None() as :

```
void None() : {} { <NONE1> | <NONE2> }
```

This will make recognition much faster. Note however, that if the choice is between two complex regular expressions, it is OK to have the choice.