

# An LALR Parser for Structured Documents Using XML

After studying the specifications of grammars and the design of an LALR parsing mechanism, the BYacc/J tool should be applied to generate the LALR parser that parses your XML documents to check their validity in respect of your XML DTD.

Therefore, we must learn about, how the BYacc/J parser generator works. The input of the BYacc/J is a file that gives the specification for *your* parser that should be generated. The specification file uses a certain notation, especially the so-called *Yacc grammar*. The notation can be studied by inspection and review of given examples. To install and to run BYacc/J, the references to the binaries (or the sources) and to the manual are given.

## Working with BYacc/J

The BYacc/J tool generates Java code that implements your parser as specified (Figure 1).

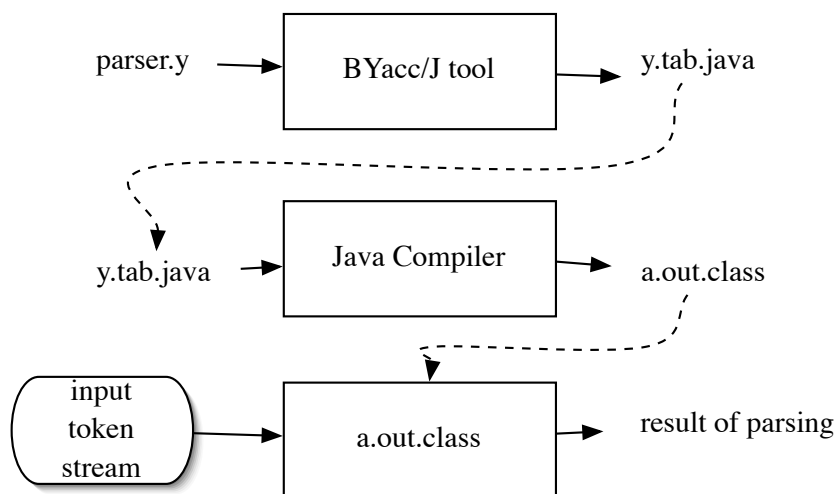


Figure 1: BYacc/J Java code generation process

In the Figure 1, the file `parser.y` includes the specification that is denoted by a certain syntax. This syntax is the BYacc/J tool inherent<sup>1</sup>. The output file, that is typically called `y.tab.java`, includes Java code. To work with the generated parser this Java code, this file `y.tab.java` must be compiled. The parser program to parse the input string has its

---

<sup>1</sup>An extensive example to specify arithmetic expressions is on the previous exercise sheet already. More syntax examples are given later.

own input. That is the token stream, which is produced by the lexer (in advance). Running the program the compiled Java program `a.out.class`, the output should be the parse of an syntactical correct input string; otherwise an error must be reported. Take care that the error messages are specific as possible.

The BYacc/J parser generator is an extension of the Yacc tool. The former Yacc tool generates C code. The BYacc/J tool generates a parser written in Java, but the specification is also given by a so-called *Yacc grammar*. The definition of the *Yacc grammar* is the same for both parser generator tools.

## On YACC specifications

In the following the structure of YACC is considered. Valid examples of YACC grammars are given for better understanding of the YACC syntax, before the installation of BYACC/J is described. To run the BYACC/J generator, it is required to give a YACC specification within a file, like `parser.y`.

### The structure of YACC specification file

The YACC specification file (using BYacc/J) is always structured as follows

```
%{  
Java declarations  
%}  
Yacc declarations  
%%  
Grammar rules  
%%  
Additional Java code
```

**Java declarations** allows to import Java code from other classes, especially from `java.io.*` and `java.util.StringTokenizer`.

**Yacc declarations** list the tokens that have been produced by the lexer. These token must be declared in correspondence to the lexer specification. The Yacc declarations also

include preparing to set preferences in respect of the certain operations, like bindings; check the specification of arithmetic expressions.

**Grammar rules** map the token stream that is generated by the lexer to the structure of the given context-free grammar.

**Additional Java code** Beside the definition of the grammar rules, the handling of the tokens must be implemented in this part of the Yacc specification file. If a token does not match with any grammar rule, an error messages has to be generated and written to the output as well. Think about implementation of an exception handling!

Be aware that

- to apply BYacc/J, the grammar must be conflict-free and the critical left-recursions must be eliminated. Thus, check the former exercises to guarantee these issues<sup>2</sup> ; and
- these grammar rules must be denoted in Yacc syntax. This means that your conflict-free context-free grammar must be rewritten.

## Examples of Yacc grammars

The conflict-free context-free grammar, that specifies your class of documents, has to be rewritten to the Yacc syntax. To get into the notation of Yacc grammars it is helpful to study more examples.

1. An example to specify arithmetic expressions using an *infix notation* has been given in the previous exercises.
2. Another Yacc grammar defining arithmetic expressions that matches with an XML DTD is given on:

<http://www.cs.wisc.edu/condor/classad/refman/node7.html>

The corresponding XML DTD is defined on

---

<sup>2</sup>To design your XML DTD, you have already developed two notations of your grammar. Now, we need the specification that is context-free again.

<http://www.cs.wisc.edu/condor/classad/refman/node8.html>

3. To learn more about characteristics of the Yacc grammar, the specification of C can be studied by reading the corresponding Yacc grammar on:

<http://www.lysator.liu.se/c/ANSI-C-grammar-y.html>

## To install and to run BYacc/J

The sources to install BYacc/J are on the Web using

<http://byaccj.sourceforge.net>

- To install BYacc/J, click on “Download“ on the top of the Web page and chose the suitable binary code.
- To run BYacc/J, check the short description on

<http://byaccj.sourceforge.net/yacc.cat>

An extensive description of the Yacc tool is available on:

<http://dinosaur.compilertools.net/yacc/index.html>

## Study and develop YACC grammars

1. Review the example of the arithmetic expressions using the infix notation and setting precedences as given on the previous exercise sheet.
2. Study the example defining arithmetic expressions by using an XML DTD.
  - (a) How does the specification matches with the XML DTD?
  - (b) What are the similarities and differences concerning the former YACC specification just defining arithmetic expressions without using an XML DTD?
3. Make *your* specification by giving a Yacc grammar that matches with your XML DTD defining structured documents as a document class.

4. Install the BYacc/J parser generator.
5. Run the BYacc/J tool to generate the requested parser and compile it with a Java compiler.