Yet Another Parser Generator
Programmer's Guide

How to: Compile a grammar

```
//First off, we call GrammarParser.ParseGrammar and supply it with the TextReader which will feed
//it with the grammar's specification. The resulting Grammar object will contain all grammar-related
//information which could have been read directly from the specification, which means it will have
//a fully set GrammarDefinition and LexerData and it will have initialized ParserData, setting it's
//SymbolNames and ProductionOutline properties.
Ilist<string> warningMessages;
Grammar grammar = GrammarParser.ParseGrammar(new StreamReader(inputFile), out warningMessages);

//We then create an instace of GrammarProcessor and call it's ComputeTables method to compute the
//action and goto tables of the parser. After this step, the ParserData property of the 'grammar'
//has all the information needed to run a parser.
GrammarProcessor processor = new GrammarProcessor();
processor.ComputeTables(grammar);

//Now we only serialize both LexerData and ParserData using this function call.
grammar.WriteRuntimeDataToFile(outputFile);
```

How to: Parse a string

```
//If the lexer/parser data is stored in a file, we first read it either through
//Grammars's or LexerData's or ParserData's static methods.
LexerData lexerData;
ParserData parserData;
Grammar.ReadRuntimeDataFromFile(parserFile, out lexerData, out parserData);

//We then use the data we have to initialize both the lexer and the parser.
Lexer lexer = new Lexer(lexerData);
Parser parser = new Parser(parserData);

//Any time we want the lexer to start scanning a new string, we simply set
//it's SourceString property.
lexer.SourceString = input;

//After we have an initialized parser and a lexer that is fed with input,
//we may call the parsers's Parse method to retrieve the ParseTree.
ParseTree parseTree = parser.Parse(lexer);
```

How to: Use a custom lexer

Do the same as in How to: Compile a Grammar, but ignore the regular expression part when writing the grammar specification; just fill in the names of the terminals followed by the equals sign. This way GrammarParser.ParseGrammar will create a degenerate LexerData object, which we will ignore. You can then compute the grammar's tables using the GrammarProcessor as before and store the resulting ParserData object using, for example, ParserData.WriteToStream(Stream). Then you can write your own lexer which has to implement the ILexer interface. The lexer returns tokens in which the terminal symbol's identity is established by a numerical code. The codes for the grammar's terminal symbols range from 1 to the number of terminals defined. The terminals are numbered in the same order in which they were defined in the grammar specification. After the lexer has scanned all of it's input, it should return a token with symbol code 0 and set it's HasTokens property to false.

For more detailed information on the workings of the API, check **yapg_api_doc.chm**, the commentated source code or try browsing the classes with IntelliSense.