FREE eBook

LEARNING ANTLR

Free unaffiliated eBook created from
Stack Overflow contributors.

#antlr

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: antlr

It is an unofficial and free ANTLR ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ANTLR.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with ANTLR

Remarks

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees.

- Official antlr website (always points to the latest version)

Antlr Versions
Antlr is separated in two big parts, the grammar (grammar files) and the generated code files, which derive from the grammar based on target language. The antlr versions are in the format of V1.V2.V3 :

- V1: Change in V1 means that new syntax of features were introduced in grammar files
- V2: Change in V2 means that new features or major fixes were introduced in the generated files (e.g addition of new functions)
- V3: stands for bug fixes or minor improvements

Runtime Libraries and Code Generation Targets
The Antlr tool is written in Java, however it is able to generate parsers and lexers in various languages. To run the parser and lexer you will also need having the runtime library of antlr alongside with the parser and lexer code. The supported target language (and runtime libraries) are the following:

- Java
- C#
- Python (2 and 3)
- JavaScript

Versions

Version	Release Date
2.0	1997-05-01
3.0	2011-01-19
4.0	2013-01-21
4.1	2013-07-01

Version	Release Date
4.2	2014-02-05
4.2.1	2014-03-25
4.2.2	2014-04-07
4.3	2014-06-19
4.4	2014-07-16
4.5	2015-01-23
4.5.1	2016-07-16
4.5.2	2016-01-30
4.5.3	2016-03-31
4.6	2016-12-15
4.7	2017-03-30

Examples

Hello world

A simple hello world grammar can be found here:

```
// define a grammar called Hello
grammar Hello;
r   : 'hello' ID;
ID  : [a-z]+ ;
WS  : [ \t\r\n]+ -> skip ;
```

To build this .g4 sample you can run the following command from your operating systems terminal/command-line:

```
Java -jar antlr-4.5.3-complete.jar Hello.g4

//OR if you have setup an alias or use the recommended batch file

antlr4 Hello.g4
```

Building this example should result in the following output in the Hello.g4 file directory:

1. Hello.tokens
2. HelloBaseListener.java
3. HelloLexer.java

4. HelloLexer.tokens
5. HelloListener.java
6. HelloParser.java

When using these files in your own project be sure to include the ANTLR jar file. To compile all of these files using Java, in the same operating directory or by path run the following command:

```
javac *.java
```

Read Getting started with ANTLR online: https://riptutorial.com/antlr/topic/4453/getting-started-with-antlr

Chapter 2: ANTLR Targets/Language Runtimes

Examples

Language Support

ANTLR is capable of generating parsers for a number of programming languages:

1. C# Target
2. Python Target
3. JavaScript Target
4. Java Target

By default ANTLR will generate a parser from commandline in the Java programming language :

```
Java -jar antlr-4.5.3-complete.jar yourGrammar.g4 //Will output a
    java parser
```

To change the target language you can run the following command from the OS terminal/commandline:

```
    antlr4 -Dlanguage=Python3 yourGrammar.g4
//with alias
    java -jar antlr-4.5.3-complete.jar -Dlanguage=Python3 yourGrammar.g4
//without alias
```

Rather than use the '-Dlanguage' parameter on the commandline/terminal each time to build your desired parser for a specific language you can select the target from within your .g4 grammar file by including the target within the global section:

```
options {
    language  = "CSharp";
}
//or
options {
    language="Python";
}
```

To use the generated parser output make sure you have the ANTLR runtime for the specified language :

1. CSharp runtime
2. Python 2 runtime
3. python 3 runtime

Full instructions and information on ANTLR run-times libraries

Python parser setup

After running your grammar .g4 file with ANTLR.jar you should have a number of files generated such as :

```
1.yourGrammarNameListener.py
2.yourGrammarNameParser.py
3.yourGrammarName.tokens
...
```

To use these in a python project include the Python runtime in your workspace so any application you are developing can access the ANTLR library. This can be done by extracting the runtime into your current project folder or importing it within your IDE into your project dependencies.

```
#main.py
import yourGrammarNameParser
import sys

#main method and entry point of application

def main(argv):
    """Main method calling a single debugger for an input script"""
    parser = yourGrammarNameParser
    parser.parse(argv)

if __name__ == '__main__':
    main(sys.argv)
```

This setup includes your parser and accepts input from commandline to allow processing of a file passed as a parameter.

```
#yourGrammarNameParser.py
from yourGrammarNameLexer import yourGrammarNameLexer
from yourGrammarNameListener import yourGrammarNameListener
from yourGrammarNameParser import yourGrammarNameParser
from antlr4 import *
import sys

class yourGrammarNameParser(object):
    """
    Debugger class - accepts a single input script and processes
    all subsequent requirements
    """
def __init__(self): # this method creates the class object.
    pass


#function used to parse an input file
def parse(argv):
    if len(sys.argv) > 1:
        input = FileStream(argv[1]) #read the first argument as a filestream
        lexer = yourGrammarNameLexer(input) #call your lexer
        stream = CommonTokenStream(lexer)
        parser = yourGrammarNameParser(stream)
        tree = parser.program() #start from the parser rule, however should be changed to your
entry rule for your specific grammar.
```

```
        printer = yourGrammarNameListener(tree,input)
        walker = ParseTreeWalker()
        walker.walk(printer, tree)
    else:
        print('Error : Expected a valid file')
```

These files coupled with the ANTLR runtime and your files generated from your grammar file will accept a single filename as an argument and read and parse your grammar rules.

To extend the basic functionality you should also expand on the default listener to handle relevant events for tokens that are encountered during runtime.

Read ANTLR Targets/Language Runtimes online: https://riptutorial.com/antlr/topic/3414/antlr-targets-language-runtimes

Chapter 3: Introduction to ANTLR v3

Examples

Installation and Setup

How To Install ANTLR in Eclipse

(Last tested on Indigo and ANTLR IDE 2.1.2)

1. Install Eclipse.
2. Download ANTLR complete binaries jar that includes ANTLR v2. Extract to a temp directory. Copy the antlr-n.n folder to an appropriate permanent location, for example the same folder that Eclipse is installed in.
3. Add ANTLR IDE update site to Eclipse.
    - In Eclipse, click on Help and select Install New Software.
    - Click Add… button.
    - In the Add Repository window, for Location type http://antlrv3ide.sourceforge.net/updates and type something like ANTLR IDE for the Name and click OK to get back to the Available Software window.
    - Check the box for ANTLR IDE vn.n.n and click on through until it is installed. Eclipse will probably restart.
4. Configure the ANTLR IDE.
    - In the Eclipse main window, click Window then Preferences.
    - In the left pane, expand ANTLR and select Builder.
    - In the right pane, click the Add… button.
    - In the Add ANTLR Package window, click Directory… and navigate to the location of the antlr-n.n folder and click OK.
    - Click OK to close the Add ANTLR Package window.
    - Select Code Generator in the left pane and click on Project relative folder in the right pane. Type a folder name. Examples: antlr-java or antlr-generated.
    - Select any other configuration parameters but DO NOT check –nfa or –dfa in the under General in the Building window. If checked, these will cause ANTLR errors preventing java files from being generated in the output folder.
    - Click OK to close the Preferences window.
5. Create a new Java project and enable ANTLR support.
    - From the Eclipse main window, go to File, New, Java Project. Click Next, type a project name and click Finish.
    - To enable ANTLR support for the project, in the Package Explorer window (left pane) right-click the project just created and select Configure, Convert to ANTLR project.
    - Add the ANTLR complete jar file to the project: right-click the project and select Properties, Java Build Path, click Add External JARs…, browse to the ANTLR jar file, select it, and click OK. Click OK to close the project Properties window.
6. Create an ANTLR grammar.

- Create a new ANTLR grammar: right-click the src folder of the project, then File, New, Other, expand ANTLR and select Combined Grammar. Click Next, type grammar name, select a Language option, and click Finish.
- A ".g" file is created with the options selected and a blank rule. Add the options language=Java, @header, @lexer::header, and @members statements at the top (see example). Auto completion is the easiest way to add these (press CTRL-space to bring up auto-completion list).

7. Save the grammar.
    - When saved, a folder containing generated Java code for the grammar should appear in the Project Explorer. If it does not, make sure the –nfa or –dfa options are not checked in ANTLR Preferences under General in the Building window (Step 4g). [Confirm if these needed: check CLASSPATH environment variable points to the Java7 that matches your Eclipse install (32 or 64 bits) and Windows Path environment variable had Java7 SDK.]
    - To avoid "cannot be resolved to a type" Java errors, right-click the folder containing generated Java code, then Build Path, Use as a Source Folder.

SAMPLE COMBINED GRAMMAR

```
grammar test; //must match filename.g

options {
    language = Java;
}

@header { //parser
    package pkgName; //optional
    import java.<whatever you need>.*;
}

@members { //parser
    // java code here
}

@lexer::header { //lexer
    package pkgName; //optional
    import java.<whatever you need>.*;
}

@lexer::members {
    // java code here
}
/*------------------------------------------------------------------
 * PARSER RULES (convention is all lowercase)
 *------------------------------------------------------------------*/
parserule: LEXRULE;

/*------------------------------------------------------------------
 * LEXER RULES (convention is all uppercase)
 *------------------------------------------------------------------*/
LEXRULE: 'a'..'z';
```

Read Introduction to ANTLR v3 online: https://riptutorial.com/antlr/topic/6629/introduction-to-antlr-v3

Chapter 4: Introduction to ANTLR v4

Remarks

ANTLR v4 is a powerful tool used for building new programming languages and processing/translating structured text or binary files. ANTLR uses a grammar you create to generate a parser which can build and traverse a parse tree (or abstract syntax tree, AST). The parser consists of output files in a target language that you specify. ANTLR v4 supports several targets including: Java, C#, JavaScript, Python2, and Python3. Support for C++ is being worked on. For working in GUI IDEs, there are plug-ins for Visual Studio, Intellij, NetBeans, and Eclipse.

For general information, visit the ANTLR website. To get serious about ANTLR, check out the highly recommended book written by Terrence Parr (the guy who created ANTLR) The Definitive ANTLR 4 Reference.

Significant Version Info

- 4.5: 01/22/15 - Added JavaScript target and upgraded C# target. 4.5 Release Notes
- 4.4: 07/16/14 - Added Python2 and Python3 as targets. 4.4 Release Notes
- 4.3: 06/18/14 - Major bug fixes; prepared for adding new targets. 4.3 Release Notes
- 4.2: 02/04/14 - Improved syntax for selecting/matching parse trees. 4.2 Release Notes
- 4.1: 06/30/13 - Improved parsing performance; export ASTs to PNG. 4.1 Release Notes
- 4.0: 01/21/13 - Initial release.

Examples

Installing for Command Line Use

ANTLR is distributed as a Java Jar file It can be downloaded here. As ANTLR is compiled as a jar file it subsequently requires the Java runtime environment to operate, if you do not have It can be downloaded here.

Once the ANTLR JAR file has been downloaded you can run ANTLR from the command line in the same way as any other JAR file:

```
Java -jar antlr-4.5.3-complete.jar
```

(Assuming you are operating in the same directory as the antlr-4.5.3-complete.jar file).

This should output something similar to this :

```
ANTLR Parser Generator  Version 4.5.3
 -o ___              specify output directory where all output is generated
 -lib ___            specify location of grammars, tokens files
 -atn                generate rule augmented transition network diagrams
```

```
-encoding ___        specify grammar file encoding; e.g., euc-jp
-message-format ___  specify output style for messages in antlr, gnu, vs2005
-long-messages       show exception details when available for errors and warnings
-listener            generate parse tree listener (default)
-no-listener         don't generate parse tree listener
-visitor             generate parse tree visitor
-no-visitor          don't generate parse tree visitor (default)
-package ___         specify a package/namespace for the generated code
-depend              generate file dependencies
-D<option>=value     set/override a grammar-level option
-Werror              treat warnings as errors
-XdbgST              launch StringTemplate visualizer on generated code
-XdbgSTWait          wait for STViz to close before continuing
-Xforce-atn          use the ATN simulator for all predictions
-Xlog                dump lots of logging info to antlr-timestamp.log
```

other recommended actions for setup include:

```
     1. Add antlr4-complete.jar to CLASSPATH, either: Permanently:
Using System Properties dialog > Environment variables > Create or append to CLASSPATH
variable Temporarily, at command line: SET CLASSPATH=.;C:\Javalib\antlr4-
complete.jar;%CLASSPATH%
     3.Create batch commands for ANTLR Tool, TestRig in dir in PATH
         antlr4.bat: java org.antlr.v4.Tool %*
         grun.bat:   java org.antlr.v4.gui.TestRig %*
```

After setup you can build an application using your .g4 grammar file :

```
Java -jar antlr-4.5.3-complete.jar yourGrammar.g4
```

You can also build an application in other languages with the -Dlanguage parameter. For example to generate C# files you would do something like this:

```
java -jar antlr-4.5.3-complete.jar yourGrammar.g4 -Dlanguage=CSharp
```

See here for full list of pre-made grammar's for common programming languages.

Installing Using Build Automation tools

Download the latest version of ANTLR and extract it to a folder.

You can use also Maven, Gradle, or other build tool to depend on its runtime (the classes the generated grammars use): `org.antlr:antlr4-runtime`.

In order to automatically -as part of the build process- generate the parser in a maven project, use the Maven plugin: `org.antlr:antlr4`.

Install in Eclipse and Build Hello World

(Tested with ANTLR 4.5.3, Eclipse Neon, ANTLR 4 IDE 0.3.5, and Java 1.8)

1. Download the latest ANTLR. Make sure to get the complete ANTLR Java binaries jar. Save

to any appropriate location, for example the folder where other Java libraries are stored. It doesn't matter where, just remember the location.

2. Install the ANTLR IDE in Eclipse.

    - From the Eclipse menu, click Help and select Eclipse Marketplace.
    - In the Find: box, type antlr and click Go.
    - Click Install for ANTLR 4 IDE.
    - Click Finish in the Confirm Selected Features window.
    - If a Security Warning window pops up, click OK.
    - Restart Eclipse.

3. Work around for the "Failed to create injector…" error.

    - When accessing ANTLR 4 Preferences in Eclipse or when the environment variable HOME is not set, the following error occurs: Failed to create injector for com.github.jknack.antlr-4ide.Antlr4 for com.github.jknack.antlr-4ide.Antlr4.
    - Make sure the environment variable HOME is set. If not, set it as appropriate for your system.
    - Download Xtext 2.7.3 to the same location as antlr-n.n.n-complete.jar.
    - In Eclipse, click on Help and select Install New Software.
    - Click Add… to get to the Add Repository window.
    - Type a name, xtext 2.7.3 for example, then click on Archive…, navigate to the Xtext 2.7.3 file and select it, then click OK.
    - In the Install window, click the Select All button then click Next> twice, accept the license agreement. and click Finish.
    - Restart Eclipse.

4. Tell Eclipse/Java where ANTLR is.

    - In Eclipse, click on Window and select Preferences.
    - In the left pane, expand Java and Build Path, then select Classpath Variables.
    - In the right pane, click New…, enter a Name, and click File… and browse to your location of antlr-n.n.n-complete.jar. Click OK to get back to the Classpath Variables window.
    - Click OK to exit Preferences.

5. (Optional) Configure the ANTLR IDE generated sources directory.

    - In the Eclipse main window, click Window then Preferences.
    - In the left pane, expand ANTLR 4 and select Tool.
    - Under Options, change the directory if desired. For example, java is my target language so I use ./antlr-java.
    - Click OK to close the Preferences window.

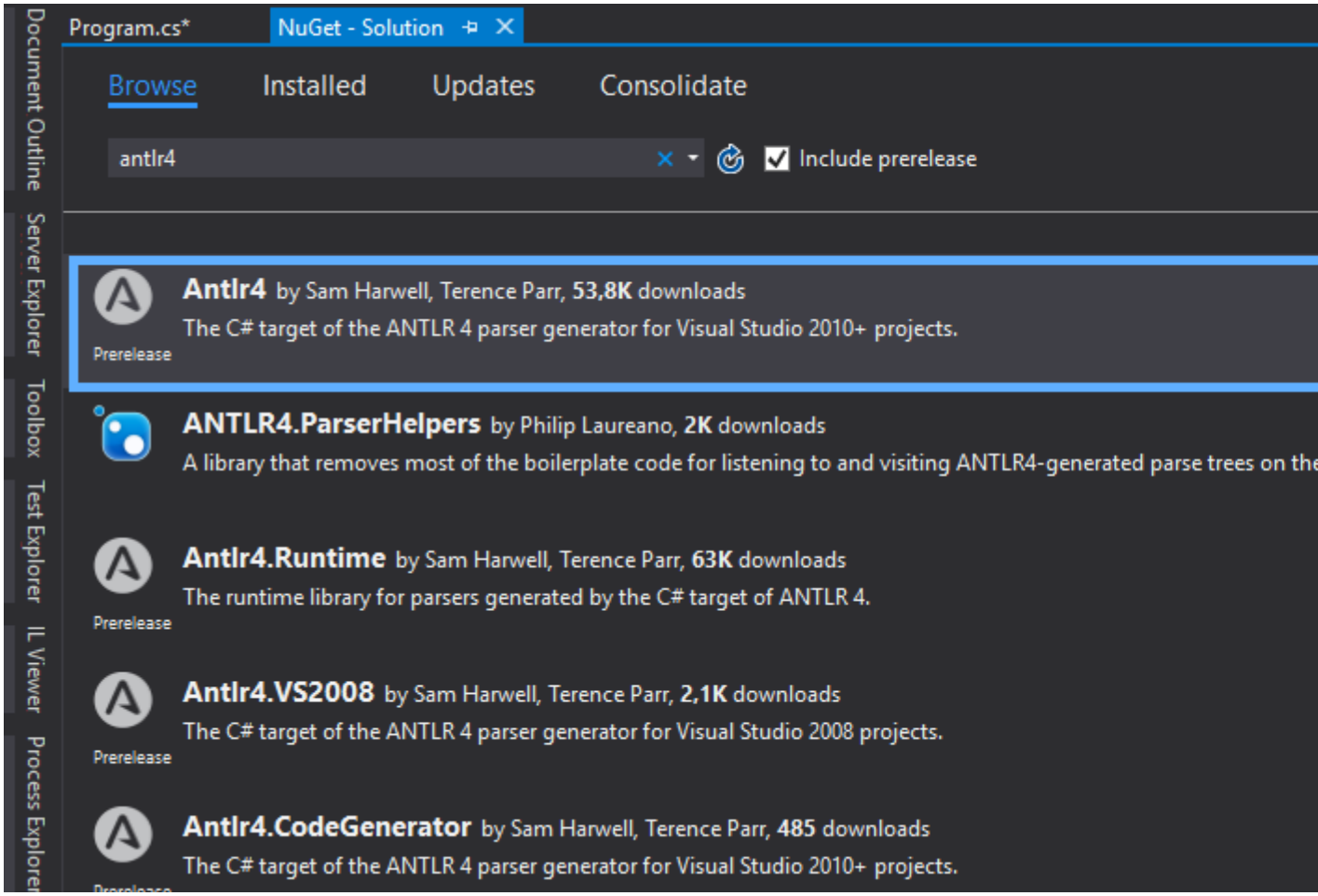
6. Create an ANTLR 4 project.

    - From the Eclipse main window, go to File, New, Project.
    - In the New Project window, expand General and select ANTLR 4 Project.

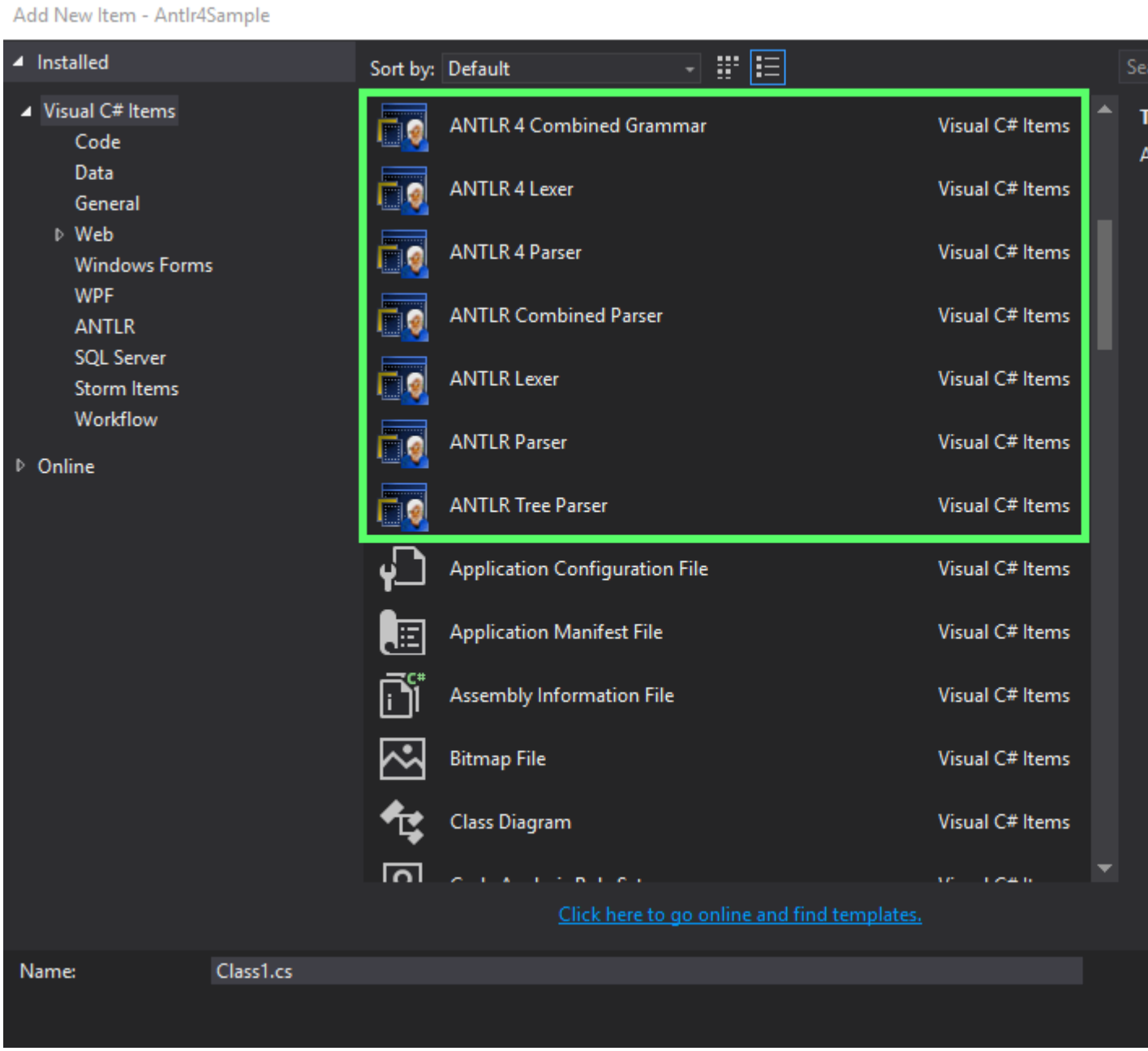
- Click Next, type a project name and click Finish.
- The default new project contains a Hello.g4 file and will automatically build the standard "Hello World" program.
- In the Package Explorer, expand the new project folder to see the g4 file and a folder named target (or the name you gave it in Step 5) containing the target source files.

Installing ANTLR in Visual Studio 2015 (using Nuget)

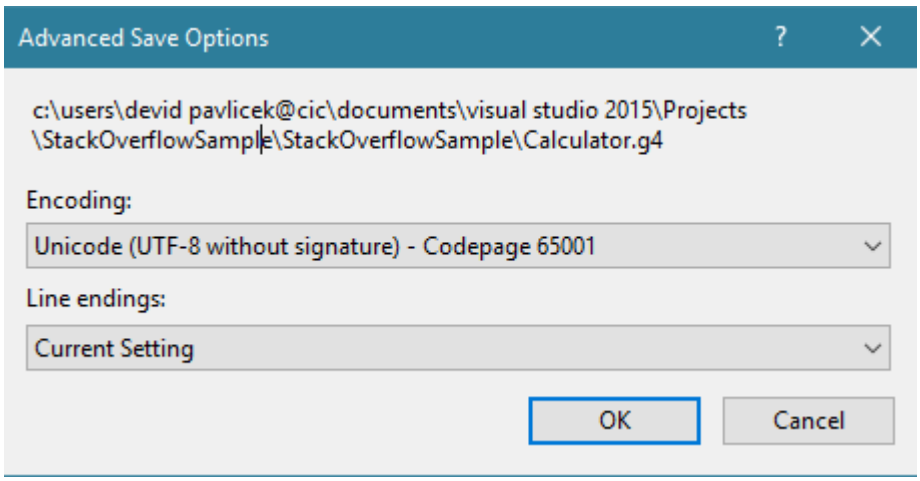
1. Open Visual Studio 2015, navigate to Tools → Extensions → Online and search for Antlr. Download the extension ANTLR Language Support (Created by Sam Harwell) and restart Visual Studio.
2. Create new Console Application Project. Right click on the Solution → Manage Nuget Packages for Solution → Browse (Tab) and search for Antlr4 and install it.



3. Add a New Item to your Project by right clicking on it. And look for ANTLR4 Templates.

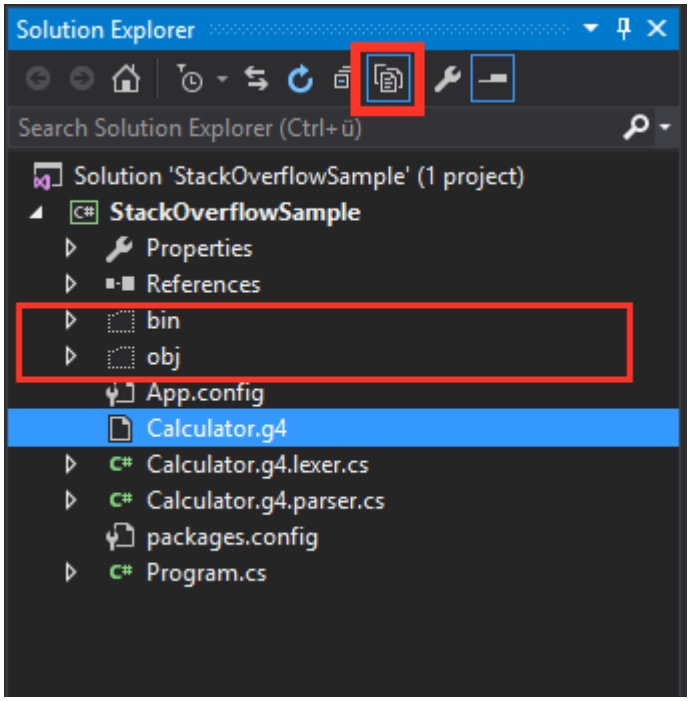


4. From your ANTLR file (ending .g4) go to File → Advance Save Options and search for Unicode (UTF-8 **without signature**) - Codepage 65001 and click OK. Thats it.

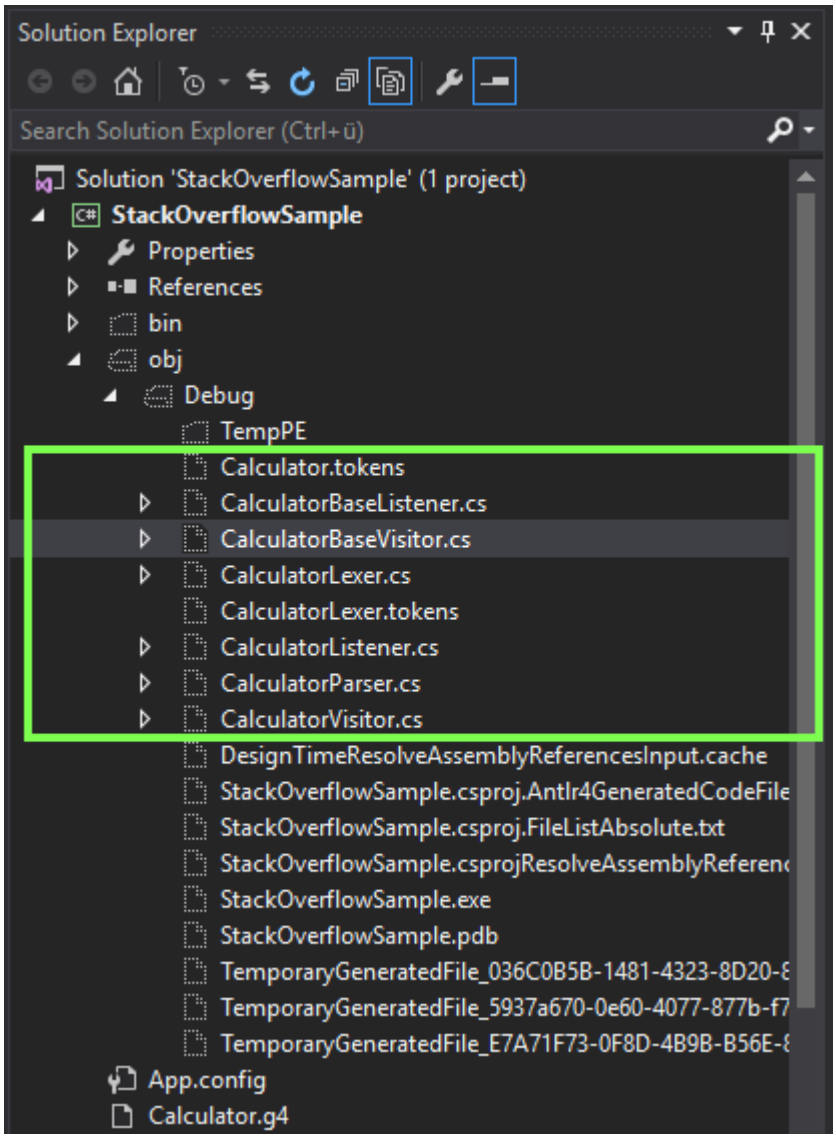


Test if everything works

- Create a ANTLR 4 Combined Grammar item and name it Calculator.g4
- Copy and Paste the Calculator source code from this Github project here: Calculator by Tom Everett
- Change grammar calculator to grammar Calculator
- On Solution Explorer → Click on Show All Files.



- Save and Run (Start) the project
- In Solution Explorer under obj folder you should see cs classes generated like the Visitor and Listener. If this is the case you succeeded. Now you can start working with ANTLR in Visual Studio 2015.



Read Introduction to ANTLR v4 online: https://riptutorial.com/antlr/topic/2856/introduction-to-antlr-v4

Chapter 5: Lexer rules in v4

Examples

Simple rules

Lexer rules define token types. Their name has to start with an uppercase letter to distinguish them from parser rules.

```
INTEGER: [0-9]+;
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;

OPEN_PAREN: '(';
CLOSE_PAREN: ')';
```

Basic syntax:

Syntax	Meaning
`A`	Match lexer rule or fragment named `A`
`A B`	Match `A` followed by `B`
`(A|B)`	Match either `A` or `B`
`'text'`	Match literal *"text"*
`A?`	Match `A` zero or one time
`A*`	Match `A` zero or more times
`A+`	Match `A` one or more times
`[A-Z0-9]`	Match one character in the defined ranges (in this example between A-Z or 0-9)
`'a'..'z'`	Alternative syntax for a character range
`~[A-Z]`	Negation of a range - match any single character *not* in the range
`.`	Match any single character

Fragments

Fragments are reusable parts of lexer rules which cannot match on their own - they need to be referenced from a lexer rule.

```
INTEGER: DIGIT+
```

```
       | '0' [Xx] HEX_DIGIT+
       ;

fragment DIGIT: [0-9];
fragment HEX_DIGIT: [0-9A-Fa-f];
```

Implicit lexer rules

When tokens like `'{'` are used in a *parser* rule, an implicit lexer rule will be created for them unless an explicit rule exists.

In other words, if you have a lexer rule:

```
OPEN_BRACE: '{';
```

Then both of these parser rules are equivalent:

```
parserRule: '{';
parserRule: OPEN_BRACE;
```

But if the `OPEN_BRACE` lexer rule is *not* defined, an implicit anonymous rule will be created. In that case, the implicit rule will be defined *as if* it were defined *before* the other rules: it will have a higher priority than other rules.

Priority rules

Several lexer rules can match the same input text. In that case, the token type will be chosen as follows:

- First, select the lexer rule which matches the *longest* input
- If the text matches an implicitly defined token (like `'{'`), use the implicit rule
- If several lexer rules match the same input length, choose the *first* one, based on definition order

---

The following combined grammar:

```
grammar LexerPriorityRulesExample;

// Parser rules

randomParserRule: 'foo'; // Implicitly declared token type

// Lexer rules

BAR: 'bar';
IDENTIFIER: [A-Za-z]+;
BAZ: 'baz';

WS: [ \t\r\n]+ -> skip;
```

Given the following input:

```
aaa foo bar baz barz
```

Will produce the following token sequence from the lexer:

```
IDENTIFIER 'foo' BAR IDENTIFIER IDENTIFIER
```

- `aaa` is of type `IDENTIFIER`

  Only the `IDENTIFIER` rule can match this token, there is no ambiguity.

- `foo` is of type `'foo'`

  The parser rule `randomParserRule` introduces the implicit `'foo'` token type, which is prioritary over the `IDENTIFIER` rule.

- `bar` is of type `BAR`

  This text matches the `BAR` rule, which is defined *before* the `IDENTIFIER` rule, and therefore has precedence.

- `baz` is of type `IDENTIFIER`

  This text matches the `BAZ` rule, but it also matches the `IDENTIFIER` rule. The latter is chosen as it is defined *before* `BAR`.

  Given the grammar, `BAZ` will *never* be able to match, as the `IDENTIFIER` rule already covers everything `BAZ` can match.

- `barz` is of type `IDENTIFIER`

  The `BAR` rule can match the first 3 characters of this string (`bar`), but the `IDENTIFIER` rule will match 4 characters. As `IDENTIFIER` matches a longer substring, it is chosen over `BAR`.

As a rule of thumb, specific rules should de defined *before* more generic rules. If a rule can only match an input which is already covered by a previously defined rule, it will *never* be used.

Implicitly defined rules such as `'foo'` act as if they were defined *before* all other lexer rules.

Lexer commands

A lexer rule can have associated *commands*:

```
WHITESPACE: [ \r\n] -> skip;
```

Commands are defined after a `->` at the end of the rule.

- `skip`: Skips the matched text, no token will be emited
- `channel(n)`: Emits the token on a different channel

- `type(n)`: Changes the emitted token type
- `mode(n)`, `pushMode(n)`, `popMode`, `more`: Controls lexer modes

Actions and semantic predicates

A lexer action is a block of arbitrary code in the target language surrounded by `{...}`, which is executed during matching:

```
IDENTIFIER: [A-Z]+ { log("matched rule"); };
```

A semantic predicate is a block of arbitrary code in the target language surrounded by `{...}?`, which evaluates to a boolean value. If the returned value is false, the lexer rule is skipped.

```
IDENTIFIER: [A-Z]+ { identifierIsValid() }?;
```

Semantic predicates should be defined at the end of the rule whenever possible for performance reasons.

Read Lexer rules in v4 online: https://riptutorial.com/antlr/topic/3271/lexer-rules-in-v4

Chapter 6: Listeners

Examples

Listener Events Using Labels

Labeling the alternatives inside a rule starting with the `#` operator tells ANTLR to generate listener methods for each label corresponding to the alternative.

By specifying a label for each alternative in the following rule:

```
// Rule
type : int     #typeInt
     | short   #typeShort
     | long    #typeLong
     | string  #typeString
     ;

// Tokens
int : 'int' ;
short : 'short' ;
long : 'long' ;
string : 'string' ;
```

Will generate the following methods in the generated interface that extends `ParseTreeListener`:

```
public void enterTypeInt(TypeShortContext ctx);
public void enterTypeShort(TypeIntContext ctx);
public void enterTypeLong(TypeLongContext ctx);
public void enterTypeString(TypeStringContext ctx);
```

Read Listeners online: https://riptutorial.com/antlr/topic/6717/listeners

Chapter 7: TestRig / grun

Examples

Setup TestRig

ANTLR contains a testing tool in its runtime library, this tool can be used to display information detailing how the parsing is performed to match input against defined rules in your grammar file.

To use this tool contained within the ANTLR jar file you should setup your systems classpath to allow access to both the ANTLR tool and the runtime library :

```
export CLASSPATH=".:/usr/local/lib/antlr-4.5.3-complete.jar:$CLASSPATH"
```

> Note: Ensure the Dot precedes any path to ensure the java virtual machine wont see classes in your current working directory.

Alises can be used on Linux/MAC/Unix to simplify commands used:

```
alias antlr4='java -jar /usr/local/lib/antlr-4.5.3-complete.jar'
//or any directory where your jar is located
```

Note setup on windows for aliases and classpath setup may be more complicated, see here for more comprehensive details.

Accessing TestRig

Once you have setup your alias you can setup TestRig in the following way, again using an alias is recommended as reduces the amount of time required to perform the action:

```
alias grun='java org.antlr.v4.runtime.misc.TestRig'
```

If you do not wish to setup an alias on windows you can access TestRig by running the following command in the same location as your ANTLR jar directory:

```
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.runtime.misc.TestRig
//or
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig
```

To run TestRig on your grammar you can pass the parameters in for your grammar like this :

```
grun yourGrammar yourRule -tree //using the setup alias
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig yourGrammar YourRule -tree //on
windows with no alias
java -cp .;antlr.4.5.3-complete.jar org.antlr.v4.gui.TestRig yourGrammar Hello r -tree
//Windows with the grammar Hello.g4 starting from the rule 'r'.
```

Build Grammar with Visual Parse Tree

Specifying the `-gui` command line option when running an ANTLR grammar in the test rig will result in a window popping up with a visual representation of the parse tree. For example:

Given the following grammar:

JSON.g4

```
/** Taken from "The Definitive ANTLR 4 Reference" by Terence Parr */

// Derived from http://json.org
grammar JSON;

json
   : value
   ;

object
   : '{' pair (',' pair)* '}'
   | '{' '}'
   ;

pair
   : STRING ':' value
   ;

array
   : '[' value (',' value)* ']'
   | '[' ']'
   ;

value
   : STRING
   | NUMBER
   | object
   | array
   | 'true'
   | 'false'
   | 'null'
   ;


STRING
   : '"' (ESC | ~ ["\\])* '"'
   ;
fragment ESC
   : '\\' (["\\/bfnrt] | UNICODE)
   ;
fragment UNICODE
   : 'u' HEX HEX HEX HEX
   ;
fragment HEX
   : [0-9a-fA-F]
   ;
NUMBER
   : '-'? INT '.' [0-9] + EXP? | '-'? INT EXP | '-'? INT
   ;
fragment INT
```

```
   : '0' | [1-9] [0-9]*
   ;
// no leading zeros
fragment EXP
   : [Ee] [+\-]? INT
   ;
// \- since - means "range" inside [...]
WS
   : [ \t\n\r] + -> skip
   ;
```

Given the following JSON file:

example.json

```
{
  "name": "John Doe",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

The following syntax command line syntax:

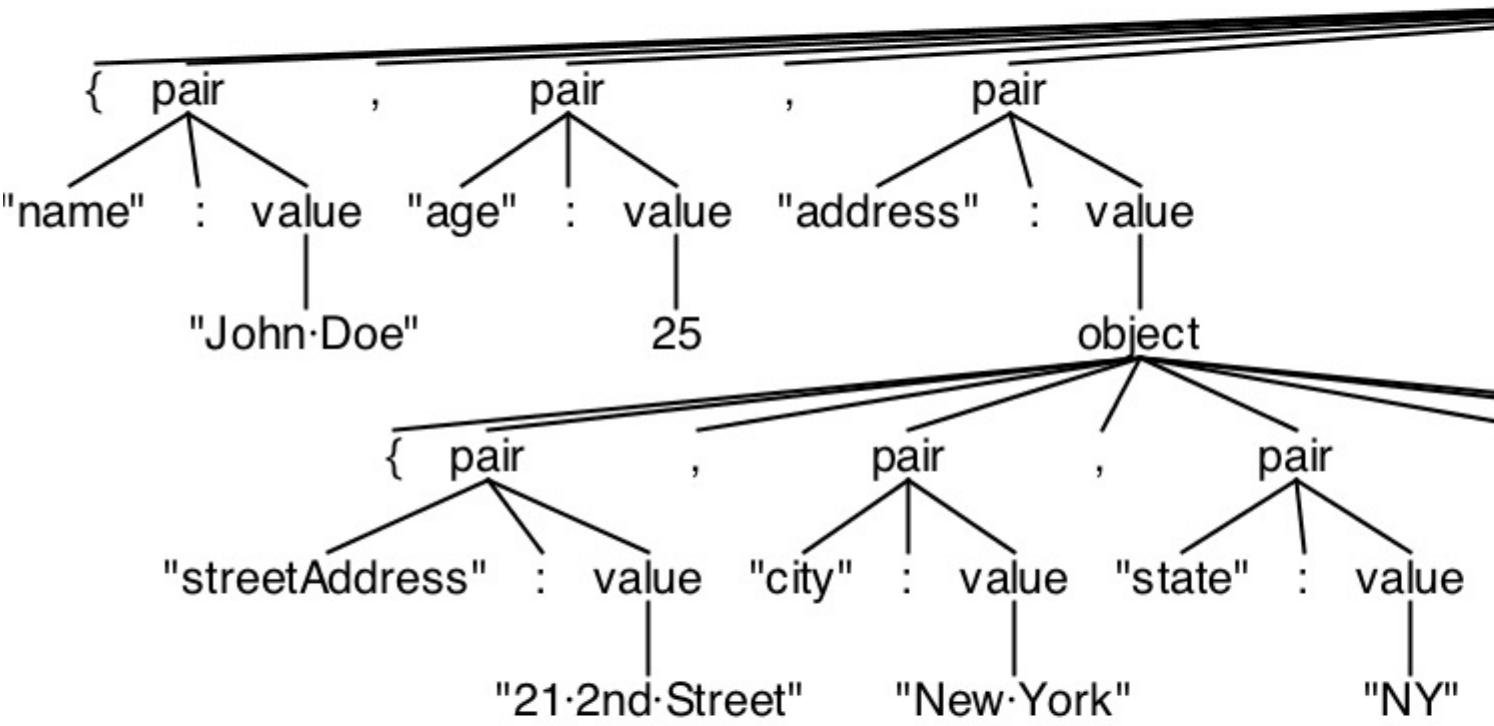
```
export CLASSPATH=".:/usr/local/lib/antlr-4.0-complete.jar:$CLASSPATH"

alias antlr4='java -jar /usr/local/lib/antlr-4.0-complete.jar'

alias grun='java org.antlr.v4.runtime.misc.TestRig'

antlr4 -o . -lib . -no-listener -no-visitor  JSON.g4; javac *.java; grun JSON json -gui
example.json
```

will result in the generated **.java** & **.tokens** files, as well as the compiled **.class** files:

```
JSON.g4                         JSONLexer.class                JSONListener.java
JSONParser$PairContext.class    JSON.tokens                    JSONLexer.java
JSONParser$ArrayContext.class   JSONParser$ValueContext.class  JSONBaseListener.class
JSONLexer.tokens                JSONParser$JsonContext.class   JSONParser.class
JSONBaseListener.java           JSONListener.class
```

```
JSONParser$ObjectContext.class JSONParser.java
```

and the following parse tree:



Read TestRig / grun online: https://riptutorial.com/antlr/topic/3270/testrig---grun

Chapter 8: Visitors

Introduction

What is the difference between a listener and a visitor? The difference between listener and visitor mechanisms is listener methods are called by the ANTLR-provided walker object, whereas visitor methods must walk their children with explicit visit calls. Forgetting to invoke visit() on a node's children means those subtrees don't get visited. In visitor we have the ability to tree walking while in listener you are only reacting to the tree walker.

Examples

Example

Grammar Example (Expr.g4)

```
grammar Expr;
prog:    (expr NEWLINE)* ;
expr:    expr ('*'|'/') expr

    |    expr ('+'|'-') expr
    |    INT
    |    '(' expr ')'
    ;
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

Generating the visitor

To generate a Visitor, or to disable a visitor for your grammar you use the following flags:

```
 -visitor           generate parse tree visitor
 -no-visitor        don't generate parse tree visitor (default)
```

The commandline/terminal command to build your grammar with a visitor will be formatted as shown below, with respect to flag chosen and possible aliases:

```
java - jar antlr-4.5.3-complete.jar Expr.g4 -visitor
java - jar antlr-4.5.3-complete.jar Expr.g4 -no-visitor
```

The output will be a parser/lexer with a visitor or no visitor respectively.

Output The output will be **ExprBaseVisitor.java** and **ExprVisitor.java** for this example. These are the relevant java files for you to implement visitor functionality. It is often ideal to create a new class and extend the ExprBaseVisitor to implement new visitor functionality for each method.

```
// Generated from Expr.g4 by ANTLR 4.5.3
```

```
import org.antlr.v4.runtime.tree.AbstractParseTreeVisitor;

/**
 * This class provides an empty implementation of {@link ExprVisitor},
 * which can be extended to create a visitor which only needs to handle a subset
 * of the available methods.
 *
 * @param <T> The return type of the visit operation. Use {@link Void} for
 * operations with no return type.
 */
public class ExprBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements ExprVisitor<T>
{
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitProg(ExprParser.ProgContext ctx) { return visitChildren(ctx); }
    /**
     * {@inheritDoc}
     *
     * <p>The default implementation returns the result of calling
     * {@link #visitChildren} on {@code ctx}.</p>
     */
    @Override public T visitExpr(ExprParser.ExprContext ctx) { return visitChildren(ctx); }
}
```

Read Visitors online: https://riptutorial.com/antlr/topic/8211/visitors

Credits

S. No	Chapters	Contributors
1	Getting started with ANTLR	Athafoud, cb4, Community, D3181, Gábor Bakos, KvanTTT
2	ANTLR Targets/Language Runtimes	D3181
3	Introduction to ANTLR v3	Athafoud, cb4
4	Introduction to ANTLR v4	Athafoud, cb4, Community, D3181, Devid, Gábor Bakos, GRosenberg, Lucas Trzesniewski
5	Lexer rules in v4	Athafoud, bn., Loxley, Lucas Trzesniewski
6	Listeners	bn., Lucas Trzesniewski
7	TestRig / grun	bn., D3181, Lucas Trzesniewski, Pascal Le Merrer
8	Visitors	D3181