Extracted from:

The Definitive ANTLR Reference

Building Domain-Specific Languages

This PDF file contains pages extracted from The Definitive ANTLR Reference, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragmaticprogrammer.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2007The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Contents

Acknowledgments **15**

Preface **16**
Why a Completely New Version of ANTLR? 18
Who Is This Book For? 20
What's in This Book? 20

I Introducing ANTLR and Computer Language Translation **22**

1 Getting Started with ANTLR **23**
1.1 The Big Picture 24
1.2 An A-mazing Analogy 28
1.3 Installing ANTLR 29
1.4 Executing ANTLR and Invoking Recognizers 30
1.5 ANTLRWorks Grammar Development Environment . . 32

2 The Nature of Computer Languages **36**
2.1 Generating Sentences with State Machines 37
2.2 The Requirements for Generating Complex Language . 40
2.3 The Tree Structure of Sentences 41
2.4 Enforcing Sentence Tree Structure 42
2.5 Ambiguous Languages 45
2.6 Vocabulary Symbols Are Structured Too 46
2.7 Recognizing Computer Language Sentences 50

3 A Quick Tour for the Impatient **61**
3.1 Recognizing Language Syntax 62
3.2 Using Syntax to Drive Action Execution 70
3.3 Evaluating Expressions via an AST Intermediate Form 75

II ANTLR Reference **87**

4 ANTLR Grammars **88**
4.1 Describing Languages with Formal Grammars 89
4.2 Overall ANTLR Grammar File Structure 91
4.3 Rules 96
4.4 Tokens Specification 116
4.5 Global Dynamic Attribute Scopes 116
4.6 Grammar Actions 118

5 ANTLR Grammar-Level Options **119**
5.1 language Option 121
5.2 output Option 122
5.3 backtrack Option 123
5.4 memoize Option 124
5.5 tokenVocab Option 124
5.6 rewrite Option 126
5.7 superClass Option 127
5.8 filter Option 128
5.9 ASTLabelType Option 129
5.10 TokenLabelType Option 130
5.11 k Option 131

6 Attributes and Actions **132**
6.1 Introducing Actions, Attributes, and Scopes 133
6.2 Grammar Actions 136
6.3 Token Attributes 140
6.4 Rule Attributes 143
6.5 Dynamic Attribute Scopes for Interrule Communication 150
6.6 References to Attributes within Actions 161

7 Tree Construction **164**
7.1 Proper AST Structure 165
7.2 Implementing Abstract Syntax Trees 170
7.3 Default AST Construction 172
7.4 Constructing ASTs Using Operators 176
7.5 Constructing ASTs with Rewrite Rules 179

8 Tree Grammars **193**
8.1 Moving from Parser Grammar to Tree Grammar 194
8.2 Building a Parser Grammar for the C- Language 197
8.3 Building a Tree Grammar for the C- Language 201

Click Here to purchase this book now.

9 Generating Structured Text with Templates and Grammars 208
9.1 Why Templates Are Better Than Print Statements . . . 209
9.2 Embedded Actions and Template Construction Rules . 211
9.3 A Brief Introduction to StringTemplate 215
9.4 The ANTLR StringTemplate Interface 216
9.5 Rewriters vs. Generators 219
9.6 A Java Bytecode Generator Using a Tree Grammar and Templates 221
9.7 Rewriting the Token Buffer In-Place 230
9.8 Rewriting the Token Buffer with Tree Grammars . . . 236
9.9 References to Template Expressions within Actions . . 240

10 Error Reporting and Recovery 243
10.1 A Parade of Errors 244
10.2 Enriching Error Messages during Debugging 247
10.3 Altering Recognizer Error Messages 249
10.4 Exiting the Recognizer upon First Error 253
10.5 Manually Specifying Exception Handlers 255
10.6 Errors in Lexers and Tree Parsers 256
10.7 Automatic Error Recovery Strategy 258

III Understanding Predicated-*LL(*)* Grammars 263

11 *LL(*)* Parsing 264
11.1 The Relationship between Grammars and Recognizers 265
11.2 Why You Need *LL(*)* 266
11.3 Toward *LL(*)* from *LL(k)* 268
11.4 *LL(*)* and Automatic Arbitrary Regular Lookahead . . . 270
11.5 Ambiguities and Nondeterminisms 275

12 Using Semantic and Syntactic Predicates 294
12.1 Syntactic Ambiguities with Semantic Predicates 295
12.2 Resolving Ambiguities and Nondeterminisms 308

13 Semantic Predicates 319
13.1 Resolving Non-LL(*) Conflicts 320
13.2 Gated Semantic Predicates Switching Rules Dynamically 327
13.3 Validating Semantic Predicates 329
13.4 Limitations on Semantic Predicate Expressions 330

Click Here to purchase this book now.

14 Syntactic Predicates **333**
14.1 How ANTLR Implements Syntactic Predicates 334
14.2 Using ANTLRWorks to Understand Syntactic Predicates 338
14.3 Nested Backtracking 339
14.4 Auto-backtracking 342
14.5 Memoization 345
14.6 Grammar Hazards with Syntactic Predicates 350
14.7 Issues with Actions and Syntactic Predicates 355

A Bibliography **359**

Index **361**

CLICK HERE to purchase this book now.

Acknowledgments

A researcher once told me after a talk I had given that "It was clear there was a single mind behind these tools." In reality, there are many minds behind the ideas in my language tools and research, though I'm a benevolent dictator with specific opinions about how ANTLR should work. At the least, dozens of people let me bounce ideas off them, and I get a lot of great ideas from the people on the ANTLR interest list.[1]

Concerning the ANTLR v3 tool, I want to acknowledge the following contributors for helping with the design and functional requirements: Sriram Srinivasan (Sriram had a knack for finding holes in my *LL(*)* algorithm), Loring Craymer, Monty Zukowski, John Mitchell, Ric Klaren, Jean Bovet, and Kay Roepke. Matt Benson converted all my unit tests to use JUnit and is a big help with Ant files and other goodies. Juergen Pfundt contributed the ANTLR v3 task for Ant. I sing Jean Bovet's praises every day for his wonderful ANTLRWorks grammar development environment. Next comes the troop of hardworking ANTLR language target authors, most of whom contribute ideas regularly to ANTLR:[2] Jim Idle, Michael Jordan (no not that one), Ric Klaren, Benjamin Niemann, Kunle Odutola, Kay Roepke, and Martin Traverso.

I also want to thank (then Purdue) professors Hank Dietz and Russell Quong for their support early in my career. Russell also played a key role in designing the semantic and syntactic predicates mechanism.

The following humans provided technical reviews: Mark Bednarczyk, John Mitchell, Dermot O'Neill, Karl Pfalzer, Kay Roepke, Sriram Srinivasan, Bill Venners, and Oliver Ziegermann. John Snyders, Jeff Wilcox, and Kevin Ruland deserve special attention for their amazingly detailed feedback. Finally, I want to mention my excellent development editor Susannah Davidson Pfalzer. She made this a much better book.

1. See http://www.antlr.org:8080/pipermail/antlr-interest/.
2. See http://www.antlr.org/wiki/display/ANTLR3/Code+Generation+Targets.

Preface

In August 1993, I finished school and drove my overloaded moving van to Minnesota to start working. My office mate was a curmudgeonly astrophysicist named Kevin, who has since become a good friend. Kevin has told me on multiple occasions that only physicists do real work and that programmers merely support physicists. Because all I do is build language tools to support programmers, I am at least two levels of indirection away from doing anything useful.[3] Now, Kevin also claims that Fortran 77 is a good enough language for anybody and, for that matter, that Fortran 66 is probably sufficient, so one might question his judgment. But, concerning my usefulness, he was right—I am fundamentally lazy and would much rather work on something that made other people productive than actually do anything useful myself. This attitude has led to my guiding principle:[4]

Why program by hand in five days what you can spend five years of your life automating?

Here's the point: The first time you encounter a problem, writing a formal, general, and automatic mechanism is expensive and is usually overkill. From then on, though, you are much faster and better at solving similar problems because of your automated tool. Building tools can also be much more fun than your real job. Now that I'm a professor, I have the luxury of avoiding real work for a living.

---

3. The irony is that, as Kevin will proudly tell you, he actually played solitaire for at least a decade instead of doing research for his boss—well, when he wasn't scowling at the other researchers, at least. He claimed to have a winning streak stretching into the many thousands, but one day Kevin was caught overwriting the game log file to erase a loss (apparently per his usual habit). A holiday was called, and much revelry ensued.

4. Even as a young boy, I was fascinated with automation. I can remember endlessly building model ships and then trying to motorize them so that they would move around automatically. Naturally, I proceeded to blow them out of the water with firecrackers and rockets, but that's a separate issue.

My passion for the last two decades has been ANTLR, ANother Tool for Language Recognition. ANTLR is a parser generator that automates the construction of language recognizers. It is a program that writes other programs.

From a formal language description, ANTLR generates a program that determines whether sentences conform to that language. By adding code snippets to the grammar, the recognizer becomes a translator. The code snippets compute output phrases based upon computations on input phrases. ANTLR is suitable for the simplest and the most complicated language recognition and translation problems. With each new release, ANTLR becomes more sophisticated and easier to use. ANTLR is extremely popular with 5,000 downloads a month and is included on all Linux and Mac OS X distributions. It is widely used because it:

- Generates human-readable code that is easy to fold into other applications
- Generates powerful recursive-descent recognizers using *LL(*)*, an extension to *LL(k)* that uses arbitrary lookahead to make decisions
- Tightly integrates StringTemplate,[5] a template engine specifically designed to generate structured text such as source code
- Has a graphical grammar development environment called ANTLRWorks[6] that can debug parsers generated in any ANTLR target language
- Is actively supported with a good project website and a high-traffic mailing list[7]
- Comes with complete source under the BSD license
- Is extremely flexible and automates or formalizes many common tasks
- Supports multiple target languages such as Java, C#, Python, Ruby, Objective-C, C, and C++

Perhaps most importantly, ANTLR is much easier to understand and use than many other parser generators. It generates essentially what you would write by hand when building a recognizer and uses technology that mimics how your brain generates and recognizes language (see Chapter 2, *The Nature of Computer Languages*, on page 36).

5. See http://www.stringtemplate.org.
6. See http://www.antlr.org/works.
7. See http://www.antlr.org:8080/pipermail/antlr-interest/.

CLICK HERE to purchase this book now.

You generate and recognize sentences by walking their implicit tree structure, from the most abstract concept at the root to the vocabulary symbols at the leaves. Each subtree represents a phrase of a sentence and maps directly to a rule in your grammar. ANTLR's grammars and resulting top-down recursive-descent recognizers thus feel very natural. ANTLR's fundamental approach dovetails your innate language process.

Why a Completely New Version of ANTLR?

For the past four years, I have been working feverishly to design and build ANTLR v3, the subject of this book. ANTLR v3 is a completely rewritten version and represents the culmination of twenty years of language research. Most ANTLR users will instantly find it familiar, but many of the details are different. ANTLR retains its strong mojo in this new version while correcting a number of deficiencies, quirks, and weaknesses of ANTLR v2 (I felt free to break backward compatibility in order to achieve this). Specifically, I didn't like the following about v2:[8]

- The v2 lexers were very slow albeit powerful.
- There were no unit tests for v2.
- The v2 code base was impenetrable. The code was never refactored to clean it up, partially for fear of breaking it without unit tests.
- The linear approximate *LL(k)* parsing strategy was a bit weak.
- Building a new language target duplicated vast swaths of logic and print statements.
- The AST construction mechanism was too informal.
- A number of common tasks were not easy (such as obtaining the text matched by a parser rule).
- It lacked the semantic predicates hoisting of ANTLR v1 (PCCTS).
- The v2 license/contributor trail was loose and made big companies afraid to use it.

ANTLR v3 is my answer to the issues in v2. ANTLR v3 has a very clean and well-organized code base with lots of unit tests. ANTLR generates extremely powerful *LL(*)* recognizers that are fast and easy to read.

8. See http://www.antlr.org/blog/antlr3/antlr2.bashing.tml for notes on what people did not like about v2. ANTLR v2 also suffered because it was designed and built while I was under the workload and stress of a new start-up (jGuru.com).

CLICK HERE to purchase this book now.

Many common tasks are now easy by default. For example, reading in some input, tweaking it, and writing it back out while preserving whitespace is easy. ANTLR v3 also reintroduces semantic predicates hoisting. ANTLR's license is now BSD, and all contributors must sign a "certificate of origin."[9] ANTLR v3 provides significant functionality beyond v2 as well:

- Powerful *LL(*)* parsing strategy that supports more natural grammars and makes it easier to build them
- Auto-backtracking mode that shuts off all grammar analysis warnings, forcing the generated parser to simply figure things out at runtime
- Partial parsing result memoization to guarantee linear time complexity during backtracking at the cost of some memory
- Jean Bovet's ANTLRWorks GUI grammar development environment
- StringTemplate template engine integration that makes generating structured text such as source code easy
- Formal AST construction rules that map input grammar alternatives to tree grammar fragments, making actions that manually construct ASTs no longer necessary
- Dynamically scoped attributes that allow distant rules to communicate
- Improved error reporting and recovery for generated recognizers
- Truly retargetable code generator; building a new target is a matter of defining StringTemplate templates that tell ANTLR how to generate grammar elements such as rule and token references

This book also provides a serious advantage to v3 over v2. Professionally edited and complete documentation is a big deal to developers. You can find more information about the history of ANTLR and its contributions to parsing theory on the ANTLR website.[10,11]

Look for *Improved in v3* and *New in v3* notes in the margin that highlight improvements or additions to v2.

---

9. See http://www.antlr.org/license.html.
10. See http://www.antlr.org/history.html.
11. See http://www.antlr.org/contributions.html.

CLICK HERE to purchase this book now.

Who Is This Book For?

The primary audience for this book is the practicing software developer, though it is suitable for junior and senior computer science undergraduates. This book is specifically targeted at any programmer interested in learning to use ANTLR to build interpreters and translators for domain-specific languages. Beginners and experts alike will need this book to use ANTLR v3 effectively. For the most part, the level of discussion is accessible to the average programmer. Portions of Part III, however, require some language experience to fully appreciate. Although the examples in this book are written in Java, their substance applies equally well to the other language targets such as C, C++, Objective-C, Python, C#, and so on. Readers should know Java to get the most out of the book.

What's in This Book?

This book is the best, most complete source of information on ANTLR v3 that you'll find anywhere. The free, online documentation provides enough to learn the basic grammar syntax and semantics but doesn't explain ANTLR concepts in detail. This book helps you get the most out of ANTLR and is required reading to become an advanced user. In particular, Part III provides the only thorough explanation available anywhere of ANTLR's *LL(*)* parsing strategy.

This book is organized as follows. Part I introduces ANTLR, describes how the nature of computer languages dictates the nature of language recognizers, and provides a complete calculator example. Part II is the main reference section and provides all the details you'll need to build large and complex grammars and translators. Part III treks through ANTLR's predicated-*LL(*)* parsing strategy and explains the grammar analysis errors you might encounter. Predicated-*LL(*)* is a totally new parsing strategy, and Part III is essentially the only written documentation you'll find for it. You'll need to be familiar with the contents in order to build complicated translators.

Readers who are totally new to grammars and language tools should follow the chapter sequence in Part I as is. Chapter 1, *Getting Started with ANTLR*, on page 23 will familiarize you with ANTLR's basic idea; Chapter 2, *The Nature of Computer Languages*, on page 36 gets you ready to study grammars more formally in Part II; and Chapter 3, *A Quick Tour for the Impatient*, on page 61 gives your brain something

Click Here to purchase this book now.

concrete to consider. Familiarize yourself with the ANTLR details in Part II, but I suggest trying to modify an existing grammar as soon as you can. After you become comfortable with ANTLR's functionality, you can attempt your own translator from scratch. When you get grammar analysis errors from ANTLR that you don't understand, then you need to dive into Part III to learn more about *LL(*)*.

Those readers familiar with ANTLR v2 should probably skip directly to Chapter 3, *A Quick Tour for the Impatient*, on page 61 to figure out how v3 differs. Chapter 4, *ANTLR Grammars*, on page 88 is also a good place to look for features that v3 changes or improves on.

If you are familiar with an older tool, such as YACC [Joh79], I recommend starting from the beginning of the book as if you were totally new to grammars and language tools. If you're used to JavaCC[12] or another top-down parser generator, you can probably skip Chapter 2, *The Nature of Computer Languages*, on page 36, though it is one of my favorite chapters.

I hope you enjoy this book and ANTLR v3 as much as I have enjoyed writing them!

Terence Parr
March 2007
University of San Francisco

12. See https://javacc.dev.java.net.

CLICK HERE to purchase this book now.

Pragmatic Projects

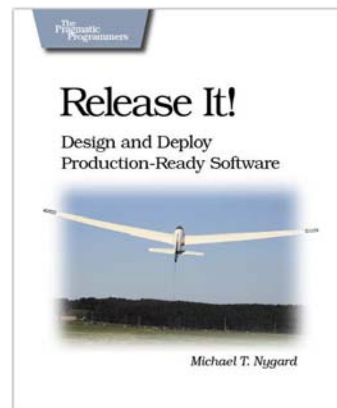
Your application is feature complete, but is it ready for the real world? See how to design and deploy production-ready software and *Release It!*.

Have you ever noticed that project retrospectives feel too little, too late? What you need to do is start having *Agile Retrospectives*.

Release It!

Whether it's in Java, .NET, or Ruby on Rails, getting your application ready to ship is only half the battle. Did you design your system to survive a sudden rush of visitors from Digg or Slashdot? Or an influx of real world customers from 100 different countries? Are you ready for a world filled with flakey networks, tangled databases, and impatient users?

If you're a developer and don't want to be on call at 3AM for the rest of your life, this book will help.

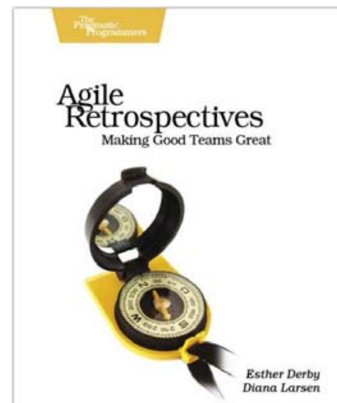


Design and Deploy Production-Ready Software
Michael T. Nygard
(368 pages) ISBN: 0-9787392-1-3. $34.95
http://pragmaticprogrammer.com/titles/mnee

Agile Retrospectives

Mine the experience of your software development team continually throughout the life of the project. Rather than waiting until the end of the project—as with a traditional retrospective, when it's too late to help—agile retrospectives help you adjust to change *today*.

The tools and recipes in this book will help you uncover and solve hidden (and not-so-hidden) problems with your technology, your methodology, and those difficult "people issues" on your team.



Agile Retrospectives: Making Good Teams Great
Esther Derby and Diana Larsen
(170 pages) ISBN: 0-9776166-4-9. $29.95
http://pragmaticprogrammer.com/titles/dlret

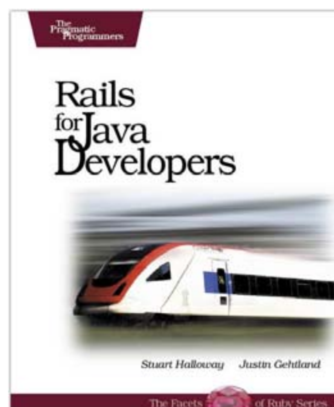
Rails and More

If you know Java, and are curious about Ruby on Rails, you don't have to start from scratch. Read *Rails for Java Developers* and get a head start on this exciting new technology.

And whatever language you use, you'll need a good text editor, too. On the Mac, we recommend TextMate.

Rails for Java Developers

Enterprise Java developers already have most of the skills needed to create Rails applications. They just need a guide which shows how their Java knowledge maps to the Rails world. That's what this book does. It covers: • The Ruby language • Building MVC Applications • Unit and Functional Testing • Security • Project Automation • Configuration • Web Services This book is the fast track for Java programmers who are learning or evaluating Ruby on Rails.



Rails for Java Developers
Stuart Halloway and Justin Gehtland
(300 pages) ISBN: 0-9776166-9-X. $34.95
http://pragmaticprogrammer.com/titles/fr_r4j

TextMate

If you're coding Ruby or Rails on a Mac, then you owe it to yourself to get the TextMate editor. And, once you're using TextMate, you owe it to yourself to pick up this book. It's packed with information which will help you automate all your editing tasks, saving you time to concentrate on the important stuff. Use snippets to insert boilerplate code and refactorings to move stuff around. Learn how to write your own extensions to customize it to the way you work.

TextMate: Power Editing for the Mac
James Edward Gray II
(200 pages) ISBN: 0-9787392-3-X. $29.95
http://pragmaticprogrammer.com/titles/textmate

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

The Definitive ANTLR Reference
http://pragmaticprogrammer.com/titles/tpantlr
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragmaticprogrammer.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragmaticprogrammer.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragmaticprogrammer.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/tpantlr.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	orders@pragmaticprogrammer.com
Non-English Versions:	translations@pragmaticprogrammer.com
Pragmatic Teaching:	academic@pragmaticprogrammer.com
Author Proposals:	proposals@pragmaticprogrammer.com