Conversion Tools - Feature #1757

update ANTLR to latest version

10/30/2012 07:03 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Conversion Tools - Feature #3883: eclipse plug for developing 4GL ...	**New**
Related to Conversion Tools - Feature #6319: IntelliJ plugin	**New**
Related to Conversion Tools - Feature #3882: changes to allow the front-end t...	**New**
Related to Conversion Tools - Support #6409: generate "railroad style" syntax...	**New**

History

#1 - 10/30/2012 07:12 AM - Greg Shah

The primary work is to change all the grammars (preprocessor, embedded 4GL, progress lexer/parser, trpl if it still exists). Any Java classes (e.g. AST implementations) that are dependent, must be udpated. The jar is updated and any build issues must be addressed.

#2 - 10/31/2012 01:15 PM - Greg Shah

- Target version set to Code Improvements

#3 - 11/16/2016 12:46 PM - Greg Shah

- Target version deleted (Code Improvements)

#4 - 05/01/2019 01:18 PM - Greg Shah

There is a suggestion that moving to ANTLRv4 will improve performance.

On the other hand, we know that tree building is massively different in v3 and I have heard that they are de-emphasizing actions (which we rely upon extensively). This will not be a trivial task.

#5 - 03/29/2022 03:34 PM - Greg Shah

ANTLR v3 is dead at this point and was replaced by ANTLR v4. Whether it makes sense or not to move to ANTLR v4 is not obvious.

These are useful guides:

https://tomassetti.me/migrating-from-antlr2-to-antlr4/
https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687070/Migrating+from+ANTLR+2+to+ANTLR+3#MigratingfromANTLR2toANTLR3-ChangesinANTLRSyntax

Please note that ANTLR v4 does not build an AST (an abstract representation of the syntax), it builds a concrete "parse tree" which is the exact grammar structure. Building an AST (which is needed for a transformation of the scale of FWD) is something we would have to do manually.

Actions are discouraged as "advanced features" because the authors want to encourage using the same grammar in different target languages. We will ignore this advice since a different target is not needed and (more importantly) we cannot parse 4GL without actions.

These two areas will make ANTLR v4 a poor match to our needs. Upgrading the ANTLR version may not be a good idea.

#6 - 02/10/2023 09:08 AM - Greg Shah

- Related to Feature #3883: eclipse plug for developing 4GL code using FWD including editing, syntax checks, running conversion added

#7 - 02/10/2023 09:08 AM - Greg Shah

- Related to Feature #6319: IntelliJ plugin added

#8 - 02/10/2023 09:24 AM - Greg Shah

- Estimated time deleted (120.00)

#9 - 02/10/2023 09:33 AM - Greg Shah

Our IDE support will need features such as auto-completion and syntax checking. These features are not suited to batch parsing runs where the parser runs for an entire input and then reports its results. Instead, the design must be able to answer questions like "Given a current cursor position and a list of edits to this source code, what are the possible valid completions to display for the user?".

That idea is discussed in relation to ANTLR, in these articles:

Implementing Code-Completion for VS Code with ANTLR
Universal Code Completion using ANTLR3
How to Implement ANTLR4 Autocomplete
Building autocompletion for an editor based on ANTLR
Java autosuggest engine for ANTLR4 grammars

Older ANTLR versions (v3 which is far beyond our current version) do not easily support this kind of processing. ANTLRv4 exposes the meta-model of the grammar itself in a way that allows these kinds of dynamic questions to be answered. It also has a far superior error handling approach that offers much more customization capabilities. This is a potentially good reason to move to ANTLRv4, though the cost is going to be high because of all the capabilities they removed along the way (actions, AST building vs parse trees...).

#10 - 02/16/2023 08:31 AM - Greg Shah

Areas that will cause some problems in ANTLRv4:

- Actions in lexers have been removed except for at the end of the rule. We will have to rework our approach in several of the 4GL lexer rules (symbols, numbers/dates, strings). It should be possible but may be tricky. Semantic predicates can still be used (thank goodness).
- The Adaptive LL(*) approach means that there is no k (fixed lookahead) value. The parsing is completely dynamic instead of statically determined at parser generation time. This means it is much more dependent upon being able to execute the prediction logic of each rule. They do this by exposing (at runtime) the "ATN" (Augented Transition Network) of the grammar. This is basically a graph network of the parsing or lexing rules where you can track the current state of the parse/lex and call the prediction logic of each rule as needed. This logic was hard coded using fixed lookahead, prediction "rollup" and sometimes "backtracking" in prior ANTLR versions. Now it is dynamic at runtime. There are implementation implications:
  - Predicates must be carefully written to:
    - Avoid any side effects. (We probably already do this but it needs to be checked)
    - Be executed without access to any rule-local state or really anything that would be detected by ANTLR as "context dependent" which would mean it cannot be executed except when actually matching.
    - Be "visible". Any predicates that don't implement the following rules cannot be used for prediction because they are "hidden" (excluded) from the prediction engine.
      - This means that they must not follow an action. Any action "in between" the current parsing location (a calling rule) and a predicate will hide that predicate because the code in the action is opaque to the prediction engine, so it assumes following predicates cannot be used for prediction. Among other implications, this probably means we must not have init actions, which will be a problem.
      - They cannot reference the current token (LT(1)) or tokens relative to the current token (LT(2) or LT(3)). The reason is that when the prediction logic is used in the calling rule, the current token is potentially very different than the state of the parse at the moment that something downstream is matched. The prediction engine may be looking arbitrarily far ahead in the token stream to determine if a particular alternative is the correct match. We do this kind of thing all over the place, so rework here is needed too.

    - We need to avoid any token rewriting in init actions. We do this in multiple ways:
        - Rewriting the type on the fly based on some lookup.
        - Merging tokens together to fix lexer limitations based on parsing context.
- All AST building now needs to be done separated from the grammar. We will need to build our own AST implementation, using the existing ANTLRv2 ASTs has negative value since they are not an efficient implementation. I think the generated listeners/visitors is not a nice implementation. This is much more work for us and I fear it is going to be some very repetitive logic that used to be very small/clean in the grammar.
- It may be slower (at runtime) than older ANTLR versions. I'm not sure about this, but several sources suggested this could be the case.

I will expand this list as I find any other issues.

Some very good things about ANTLRv4:

- Exposes the ATN for our own usage. This will be critical for implementing proper IDE support and for syntax checking. IDE support in particular requires us to be highly dynamic in our parsing, which cannot be done in the old style approach.
- Exposes a much richer error handling model, with better customization options. This is also critical to the IDE support and syntax checking.
- Handles left recursive grammars (actually, only direct left recursion, we still need to avoid indirect left recursion). This will make it possible to match a range of features which we had to handle in a more awkward way before.
- The Adaptive LL(*) approach means that we no longer have problems with ambiguity or with having to bend the parser's lookahead limitations to match the4GL behavior. It is likely this will reduce the complexity of the parsing.
- The new lexer modes feature may allow us to greatly simplify:
    - Our preprocessor lexing to a single lexer with modes instead of two lexers and the ClearStream lexer switching approach.
    - Our Progress lexer which today does not match things properly when the 4GL goes into its special modes. We make up for that with rewriting in the parser. That rewriting needs to go away for prediction reasons so cleaner lexer, here we come!
- Grammar imports may be useful to share some rules. We will have to see if the limitations are a problem.
- There is better IDE tooling for writing and testing grammars in v4.
- We can generate railroad diagrams for our grammars in v4, which will be super useful for documentation. My previous approach was to migrate our parser to ANTLRv3, strip out the actions and use a 3rd party tool to generate the diagrams. It was painful and I don't want to do it again. The new approach is possible because in ANTLRv4, you design things so that the prediction logic is all visible/callable and can be exposed in the ATN. That means that a diagramming tool can be written to walk the ATN (which is just a representation of the grammar at runtime) and render it. I think this tool already exists so that is a benefit.
- I suspect that processing for IDE support and syntax checking will be easier from the parse tree than from the AST, since we literally need to provide feedback tightly coupled to the text. For our other use cases, the AST is essential bubt for the language server, it will probably work at the parse tree level.

There are some tricky parts but I do think it is the right thing to move to ANTLRv4.

#11 - 02/16/2023 09:56 AM - Greg Shah

- Related to Feature #3882: changes to allow the front-end to be used for 4GL syntax checking added

#12 - 03/03/2023 02:04 PM - Greg Shah

- Related to Support #6409: generate "railroad style" syntax diagrams for the progress parser added