Conversion Tools - Feature #2110

Feature # 1739 (New): add hints/features for better control over comment conversion

improve orphan comment processing

04/08/2013 01:05 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Greg Shah	**% Done:**	100%
Category:		**Estimated time:**	32.00 hours
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Conversion Tools - Feature #3696: implement Progress AST anti-parser	**WIP**
Related to Conversion Tools - Bug #3711: resolve remaining orphaned comments ...	**New**

History

#1 - 04/08/2013 01:07 PM - Greg Shah

1. Reduce the number of orphan comments by implementing better heuristics for matching comments to the code they reference.

2. Provide a hints mechanism to drop orphan comments. Make sure this mechanism can be used at the per file, per directory or entire project level.

#2 - 04/08/2013 01:15 PM - Greg Shah

- Estimated time set to 32.00

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

- Target version deleted (Code Improvements)

#4 - 12/08/2017 01:05 PM - Greg Shah

I think I know the cause of most (if not all) of our orphaned comment problem.

It is caused by how we deal with tokens that are "syntactic sugar". These are tokens that:

- exist only to enhance readability by humans; OR
- which perform some function during parsing of the stream of tokens but which is not needed in the final tree form because the AST structure is unambiguous.

In the parser, we match on any such tokens but we then drop them such that they do not appear in the resulting AST.

For example, at the end of most 4GL statements is a DOT token. In the parser, this is almost always matched as DOT!, the exclamation point in the grammar tells ANTLR to drop that token (that no AST node should be created for it).

The problem here is that our comment processing is based on "hidden tokens" which are accessed from the non-hidden tokens by casting the token to CommonHiddenStreamToken and then calling getHiddenBefore() or getHiddenAfter(). We currently mark WS, TILDE, COMMENT and BACKSLASH as hidden tokens when they are created in the lexer (see setupHiddenFilter()). The ANTLR generated lexer then maintains these linkages between the tokens that are returned, such that the normal matching of the parser only occurs on the non-hidden tokens, BUT one can retrieve any hidden tokens (before or after or both) from any given token. We store these in AnnotatedAst.extractTokenData() when an AST node is created. Then during AnnotatedAst.fixups(), we call createShadowNodes() to create ShadowNode instances from the hidden tokens that are linked to a given node. These shadow tokens are stored at the end of the AST file when it is persisted and they have the linkages to the nodes on their left and/or right. Using these shadow nodes, we iterate through the comments and attempt to associate each comment with the non-hidden 4GL code that is before or after it, depending on some heuristics.

Our problem happens in the parser, when we drop tokens from the tree. Any hidden before/after nodes are not processed in these cases because no AST is ever created. So any time we drop tokens (syntactic sugar), we are losing any knowledge about the linkages that exist and we may lose entire

sections of hidden nodes if they are in between two dropped tokens.

I don't have any simple fix for this. It is going to take some thought to find a solution that doesn't require extra processing in every location of the parser where we drop nodes.

To the degree that we have had some trouble with analyzing hidden tokens, I suspect this is the cause. I know that we have found some strange behavior when trying to put some expressions "back together" for reporting purposes. We also have a TODO in AnnotatedAst.createShadowNodes() which is certainly some cleanup code that was caused by this issue.

#5 - 07/25/2018 05:50 PM - Greg Shah

The solution to this seems fairly straightforward. The dropped tokens need to be patched in to the hidden channels in the TokenStreamHiddenTokenFilter. It is probably likely that we need to subclass it and access the protected data members nextMonitoredToken and lastHiddenToken to ensure the state is correct. But the key idea here is that we can double link the dropped tokens with the AST node before and the following token (which will always be LT(1)). We can call CommonHiddenStreamToken.setHiddenAfter() and CommonHiddenStreamToken.setHiddenBefore() to setup these links. From there the only other thing to ensure is that AST persistence, the brew TRPL rules and any other TRPL code that is dependent on shadow node types can suitably ignore these other hidden nodes.

#6 - 07/31/2018 05:53 PM - Greg Shah

Created branch 2110a from trunk 11275.

Checked in revision 11276 in which extends the ANTLR token filter and hidden token classes to hook the processing so that additional state can be stored. Tokens now have a link to the previous and next non-hidden tokens in the stream. The parser was changed to use these classes instead of the ANTLR version.

Next changes will made to fixup dropped tokens to be linked into the hidden token stream as if they had always been hidden.

#7 - 08/01/2018 03:53 PM - Constantin Asofiei

If you are still working with the comments, there is an issue with converting single-line comments. This:

```
// a call from within the driver's page switch
```

gets converted to this:

```
/* a call from within the driver's page swit*/
```

Not how the last two chars get lost.

#8 - 08/01/2018 04:22 PM - Greg Shah

I'll look at it.

#9 - 08/14/2018 10:24 AM - Greg Shah

Branch 2110a was rebased from trunk 11278. The latest revision is now 11281.

#10 - 08/23/2018 07:57 AM - Greg Shah

- Related to Feature #3696: implement Progress AST anti-parser added

#11 - 09/06/2018 05:19 PM - Greg Shah

Rebased 2110a from trunk 11282. The latest revision is 11292.

#12 - 09/06/2018 05:25 PM - Greg Shah

- % Done changed from 0 to 100

- Status changed from New to Test

- Assignee set to Greg Shah

Revision 11292 includes all necessary parser changes and the changes to comment processing to eliminate all orphan comments. It also resolves:

- the // "innard" truncation issue (from #2110-7)
- improves "blocking" of related comments (comments on sequential lines with no intervening non-whitespace and which have the same starting column
- (mostly) improved location calculation for comments
- better naming conversion to handle some edge cases (more malformed symbols and the inclusion of invalid Java identifier characters)

This version of the branch is going to be regression tested now.

#13 - 09/12/2018 05:42 PM - Greg Shah

Revision 11295 passed ChUI conversion regression testing (only minor whitespace changes in DMOs was found) and Hotel GUI conversion regression testing. I ran ChUI runtime regression testing in case I had missed something in the code changes, but that also passed.

Unfortunately, a conversion regression in a large GUI application was found. This was fixed in revision 11296 and testing is going to restart after a rebase.

#14 - 09/12/2018 05:42 PM - Greg Shah

Branch 2110a rebased from trunk 11283. The latest revision is now 11297.

#15 - 09/13/2018 09:38 AM - Greg Shah

Revision 11297 passed ChUI conversion regression testing, but a regression was found in a large GUI application. The fix is in revision 11298. Testing is restarting now.

#16 - 09/13/2018 11:39 PM - Greg Shah

ChUI conversion regression testing passed (runtime testing was not needed).

Hotel GUI (conversion and runtime) passed testing. I tested Swing, virtual desktop and embedded mode. BTW, trunk has regressions in Hotel GUI embedded mode. The tab changes don't always work (blank screen) and the dismissing of dialogs sometimes leaves the overlay behind above the screen below.

The large GUI application conversion passed testing. I think we need runtime testing for that case, I'll figure that out tomorrow.

Eric: Can you run conversion on the appserver code with 2110a rev 11298 and then run the automated testing suite?

#17 - 09/14/2018 12:38 PM - Greg Shah

With some of Eric's help, we tested the large GUI application. No issues related to 2110a were found. The testing was not extensive, but it was multiple scenarios and we did not see anything.

If the server-side application converts/compiles and the automated appserver testing works, then I will merge this to trunk.

#18 - 09/14/2018 12:41 PM - Greg Shah

- Related to Bug #3711: resolve remaining orphaned comments cases added

#19 - 09/16/2018 12:57 PM - Eric Faulhaber

Greg Shah wrote:

> If the server-side application converts/compiles and the automated appserver testing works, then I will merge this to trunk.

It converted, compiled, and passed automated testing.

#20 - 09/16/2018 05:38 PM - Greg Shah

- Status changed from Test to Closed

Branch 2110a was merged to trunk as revision 11284.