Conversion Tools - Bug #3353

lexer num_literal should support hexadecimal literals

10/11/2017 10:19 AM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Greg Shah	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Conversion Tools - Bug #3352: non-reserved keywords in temp-table ...	**Closed**	
Related to Conversion Tools - Bug #3354: func_class return type not recognized	**Closed**	
Related to Conversion Tools - Bug #3351: create widget assign clause doesn't ...	**Closed**	
Related to Conversion Tools - Bug #3360: method return type can have an optio...	**Closed**	**10/19/2017**
Related to Base Language - Feature #3343: implement a 4GL syntax extension an...	**Closed**	

History

#1 - 10/11/2017 10:23 AM - Greg Shah

Submitted by Neil in a forum post:

ABL allows the use of literal Hex values and the compiler correctly interprets them as integer literals. For example:

```
    IF GET-BYTE(mXMLData, 1) <> 0x3C THEN
```

is okay, but the FWD parser will throw an error:

```
     [java] line 190:34: expecting KW_THEN, found 'x3C'
     [java]     at antlr.Parser.match(Parser.java:211)
     [java]     at com.goldencode.p2j.uast.ProgressParser.then_clause(ProgressParser.java:34804)
     [java]     at com.goldencode.p2j.uast.ProgressParser.if_stmt(ProgressParser.java:29004)
     [java]     at com.goldencode.p2j.uast.ProgressParser.stmt_list(ProgressParser.java:23453)
     [java]     at com.goldencode.p2j.uast.ProgressParser.statement(ProgressParser.java:6087)
     [java]     at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:4958)
     [java]     at com.goldencode.p2j.uast.ProgressParser.block(ProgressParser.java:4678)
     [java]     at com.goldencode.p2j.uast.ProgressParser.inner_block(ProgressParser.java:5895)
     [java]     at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:4950)
     [java]     at com.goldencode.p2j.uast.ProgressParser.block(ProgressParser.java:4678)
     [java]     at com.goldencode.p2j.uast.ProgressParser.external_proc(ProgressParser.java:4605)
     [java]     at com.goldencode.p2j.uast.AstGenerator.parse(AstGenerator.java:1487)
     [java]     at com.goldencode.p2j.uast.AstGenerator.processFile(AstGenerator.java:957)
     [java]     at com.goldencode.p2j.uast.ScanDriver.lambda$scan$0(ScanDriver.java:375)
     [java]     at com.goldencode.p2j.uast.ScanDriver.scan(ScanDriver.java:410)
     [java]     at com.goldencode.p2j.uast.ScanDriver.scan(ScanDriver.java:248)
     [java]     at com.goldencode.p2j.convert.ConversionDriver.runScanDriver(ConversionDriver.java:496)
     [java]     at com.goldencode.p2j.convert.ConversionDriver.front(ConversionDriver.java:377)
     [java]     at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:2005)
     [java] line 193:49: unexpected token: x3C
     [java]     at com.goldencode.p2j.uast.ProgressParser.lvalue(ProgressParser.java:13643)
     [java]     at com.goldencode.p2j.uast.ProgressParser.primary_expr(ProgressParser.java:53111)
     [java]     at com.goldencode.p2j.uast.ProgressParser.chained_object_members(ProgressParser.java:19400)
     [java]     at com.goldencode.p2j.uast.ProgressParser.assign(ProgressParser.java:34739)
```

To prevent the parser error, change the Hex literal to an integer literal, e.g.

```
    IF GET-BYTE(mXMLData, 1) <> 60 THEN
```

#2 - 10/11/2017 12:06 PM - Greg Shah

- Subject changed from lexer num_literal should support hexidecimal literals to lexer num_literal should support hexadecimal literals

I've created a testcase to explore how it works (see testcases/uast/hex_literals.p inside the GCD network). See the comments in the testcase for the findings.

```
def var num as int64.

/*

Hexadecimal Literal syntax is not documented in the 4GL reference, but there are examples
in the enum section for OpenEdge v11.

The rules:

- there is an optional leading minus sign
- the first mandatory character must be the digit 0
- the second mandatory character must be x or X
- all other characters are optional, if just a 0x or 0X appears then the x/X is ignored and the result is 0
- if the hexadecimal characters exist, there must be no intervening whitespace
- after the X or x, there may be between 1 and 16 following hexadecimal digits (0-9,a-f,A-F)
- both even and odd numbers of digits are accepted
- there is no case sensitivity in this syntax
- no leading 0 results in a compile error "Unknown Field or Variable name - x1. (201)"
- no x or X will parse if there are no a-f or A-F digits, or it will fail with compile error "Unknown Field or
 Variable name - 01A. (201)"
- use of invalid hexadecimal digits (like an L) will fail with a compile error "Unknown Field or Variable name
 - 0x1L. (201)"
- use of more than 16 hex digits (e.g. 0x00199999999999999999999999) will yield this compile time error: "Hex
idecimal input greater than 16 digits is not supported. (14342)"
- at runtime, it seems that the actual value being stored is an unsigned 64-bit value which means that hexadec
imal literals are numbers between 0 and 18,446,744,073,709,551,615 (0xFFFFFFFFFFFFFFFF)
- negative numbers can only be represented using a unary minus prefix, -0x1 is the same as -1, -0xFFFFFFFFFFF
FFFF is -18446744073709551615
- you cannot use a unary plus prefix, +0x101 generates compile error "** Unknown Field or Variable name - +0x1
01. (201)"
- postfixing a unary minus causes compile error "** Unknown Field or Variable name - 0x01-. (201)"
- use of larger literals being assigned to an integer (32-bit size) will yield this runtime error "Value 34056
91582 too large to fit in INTEGER. Line 569 in hex_literals.p. (15747)"
- int64 is a signed 64-bit value, which means it ranges between -9,223,372,036,854,775,808 and 9,223,372,036,8
54,775,807
- attempting to assign any value too large for a signed 64-bit integer (0x8000000000000000 or larger) to an in
t64 will cause this runtime error "** Value too large for integer. (78)"
- attempting to assign any value too small for a signed 64-bit integer (-0x8000000000000001 or smaller) to an
 int64 will cause this runtime error "** Value too large for integer. (78)" (yes, the message is confusing)
- this behavior suggests that internally, the hexadecimal literal might be processed using a 128-bit signed va
lue since the actual values can be used but just not assigned (for example, message 0xFFFFFFFFFFFFFFFF works b
ut num = 0xFFFFFFFFFFFFFFFF does not work)

*/

num = 0x.
num = 0X.
num = 0x0.
num = 0x1.
num = 0x2.
num = 0x3.
num = 0x4.
num = 0x5.
num = 0x6.
num = 0x7.
num = 0x8.
num = 0x9.
num = 0xa.
num = 0xb.
```

```
num = 0xc.
num = 0xd.
num = 0xe.
num = 0xf.
num = 0X0.
num = 0X1.
num = 0X2.
num = 0X3.
num = 0X4.
num = 0X5.
num = 0X6.
num = 0X7.
num = 0X8.
num = 0X9.
num = 0XA.
num = 0XB.
num = 0XC.
num = 0XD.
num = 0XE.
num = 0XF.
num = 0xA.
num = 0xB.
num = 0xC.
num = 0xD.
num = 0xE.
num = 0xF.
num = 0Xa.
num = 0Xb.
num = 0Xc.
num = 0Xd.
num = 0Xe.
num = 0Xf.
num = 0x00.
num = 0x01.
num = 0x02.
num = 0x03.
num = 0x04.
num = 0x05.
num = 0x06.
num = 0x07.
num = 0x08.
num = 0x09.
num = 0x0a.
num = 0x0b.
num = 0x0c.
num = 0x0d.
num = 0x0e.
num = 0x0f.
num = 0x0A.
num = 0x0B.
num = 0x0C.
num = 0x0D.
num = 0x0E.
num = 0x0F.
num = 0x10.
num = 0x11.
num = 0x12.
num = 0x13.
num = 0x14.
num = 0x15.
num = 0x16.
num = 0x17.
num = 0x18.
num = 0x19.
num = 0x1a.
num = 0x1b.
num = 0x1c.
num = 0x1d.
num = 0x1e.
num = 0x1f.
num = 0x1A.
num = 0x1B.
num = 0x1C.
num = 0x1D.
num = 0x1E.
```

```
num = 0x1F.
num = 0x20.
num = 0x21.
num = 0x22.
num = 0x23.
num = 0x24.
num = 0x25.
num = 0x26.
num = 0x27.
num = 0x28.
num = 0x29.
num = 0x2a.
num = 0x2b.
num = 0x2c.
num = 0x2d.
num = 0x2e.
num = 0x2f.
num = 0x2A.
num = 0x2B.
num = 0x2C.
num = 0x2D.
num = 0x2E.
num = 0x2F.
num = 0x30.
num = 0x31.
num = 0x32.
num = 0x33.
num = 0x34.
num = 0x35.
num = 0x36.
num = 0x37.
num = 0x38.
num = 0x39.
num = 0x3a.
num = 0x3b.
num = 0x3c.
num = 0x3d.
num = 0x3e.
num = 0x3f.
num = 0x3A.
num = 0x3B.
num = 0x3C.
num = 0x3D.
num = 0x3E.
num = 0x3F.
num = 0x40.
num = 0x41.
num = 0x42.
num = 0x43.
num = 0x44.
num = 0x45.
num = 0x46.
num = 0x47.
num = 0x48.
num = 0x49.
num = 0x4a.
num = 0x4b.
num = 0x4c.
num = 0x4d.
num = 0x4e.
num = 0x4f.
num = 0x4A.
num = 0x4B.
num = 0x4C.
num = 0x4D.
num = 0x4E.
num = 0x4F.
num = 0x50.
num = 0x51.
num = 0x52.
num = 0x53.
num = 0x54.
num = 0x55.
num = 0x56.
num = 0x57.
```

```
num = 0x58.
num = 0x59.
num = 0x5a.
num = 0x5b.
num = 0x5c.
num = 0x5d.
num = 0x5e.
num = 0x5f.
num = 0x5A.
num = 0x5B.
num = 0x5C.
num = 0x5D.
num = 0x5E.
num = 0x5F.
num = 0x60.
num = 0x61.
num = 0x62.
num = 0x63.
num = 0x64.
num = 0x65.
num = 0x66.
num = 0x67.
num = 0x68.
num = 0x69.
num = 0x6a.
num = 0x6b.
num = 0x6c.
num = 0x6d.
num = 0x6e.
num = 0x6f.
num = 0x6A.
num = 0x6B.
num = 0x6C.
num = 0x6D.
num = 0x6E.
num = 0x6F.
num = 0x70.
num = 0x71.
num = 0x72.
num = 0x73.
num = 0x74.
num = 0x75.
num = 0x76.
num = 0x77.
num = 0x78.
num = 0x79.
num = 0x7a.
num = 0x7b.
num = 0x7c.
num = 0x7d.
num = 0x7e.
num = 0x7f.
num = 0x7A.
num = 0x7B.
num = 0x7C.
num = 0x7D.
num = 0x7E.
num = 0x7F.
num = 0x80.
num = 0x81.
num = 0x82.
num = 0x83.
num = 0x84.
num = 0x85.
num = 0x86.
num = 0x87.
num = 0x88.
num = 0x89.
num = 0x8a.
num = 0x8b.
num = 0x8c.
num = 0x8d.
num = 0x8e.
num = 0x8f.
num = 0x8A.
```

```
num = 0x8B.
num = 0x8C.
num = 0x8D.
num = 0x8E.
num = 0x8F.
num = 0x90.
num = 0x91.
num = 0x92.
num = 0x93.
num = 0x94.
num = 0x95.
num = 0x96.
num = 0x97.
num = 0x98.
num = 0x99.
num = 0x9a.
num = 0x9b.
num = 0x9c.
num = 0x9d.
num = 0x9e.
num = 0x9f.
num = 0x9A.
num = 0x9B.
num = 0x9C.
num = 0x9D.
num = 0x9E.
num = 0x9F.
num = 0xA0.
num = 0xA1.
num = 0xA2.
num = 0xA3.
num = 0xA4.
num = 0xA5.
num = 0xA6.
num = 0xA7.
num = 0xA8.
num = 0xA9.
num = 0xAa.
num = 0xAb.
num = 0xAc.
num = 0xAd.
num = 0xAe.
num = 0xAf.
num = 0xAA.
num = 0xAB.
num = 0xAC.
num = 0xAD.
num = 0xAE.
num = 0xAF.
num = 0xa0.
num = 0xa1.
num = 0xa2.
num = 0xa3.
num = 0xa4.
num = 0xa5.
num = 0xa6.
num = 0xa7.
num = 0xa8.
num = 0xa9.
num = 0xaa.
num = 0xab.
num = 0xac.
num = 0xad.
num = 0xae.
num = 0xaf.
num = 0xaA.
num = 0xaB.
num = 0xaC.
num = 0xaD.
num = 0xaE.
num = 0xaF.
num = 0xB0.
num = 0xB1.
num = 0xB2.
num = 0xB3.
```

```
num = 0xB4.
num = 0xB5.
num = 0xB6.
num = 0xB7.
num = 0xB8.
num = 0xB9.
num = 0xBa.
num = 0xBb.
num = 0xBc.
num = 0xBd.
num = 0xBe.
num = 0xBf.
num = 0xBA.
num = 0xBB.
num = 0xBC.
num = 0xBD.
num = 0xBE.
num = 0xBF.
num = 0xb0.
num = 0xb1.
num = 0xb2.
num = 0xb3.
num = 0xb4.
num = 0xb5.
num = 0xb6.
num = 0xb7.
num = 0xb8.
num = 0xb9.
num = 0xba.
num = 0xbb.
num = 0xbc.
num = 0xbd.
num = 0xbe.
num = 0xbf.
num = 0xbA.
num = 0xbB.
num = 0xbC.
num = 0xbD.
num = 0xbE.
num = 0xbF.
num = 0xC0.
num = 0xC1.
num = 0xC2.
num = 0xC3.
num = 0xC4.
num = 0xC5.
num = 0xC6.
num = 0xC7.
num = 0xC8.
num = 0xC9.
num = 0xCa.
num = 0xCb.
num = 0xCc.
num = 0xCd.
num = 0xCe.
num = 0xCf.
num = 0xCA.
num = 0xCB.
num = 0xCC.
num = 0xCD.
num = 0xCE.
num = 0xCF.
num = 0xc0.
num = 0xc1.
num = 0xc2.
num = 0xc3.
num = 0xc4.
num = 0xc5.
num = 0xc6.
num = 0xc7.
num = 0xc8.
num = 0xc9.
num = 0xca.
num = 0xcb.
num = 0xcc.
```

```
num = 0xcd.
num = 0xce.
num = 0xcf.
num = 0xcA.
num = 0xcB.
num = 0xcC.
num = 0xcD.
num = 0xcE.
num = 0xcF.
num = 0xD0.
num = 0xD1.
num = 0xD2.
num = 0xD3.
num = 0xD4.
num = 0xD5.
num = 0xD6.
num = 0xD7.
num = 0xD8.
num = 0xD9.
num = 0xDa.
num = 0xDb.
num = 0xDc.
num = 0xDd.
num = 0xDe.
num = 0xDf.
num = 0xDA.
num = 0xDB.
num = 0xDC.
num = 0xDD.
num = 0xDE.
num = 0xDF.
num = 0xd0.
num = 0xd1.
num = 0xd2.
num = 0xd3.
num = 0xd4.
num = 0xd5.
num = 0xd6.
num = 0xd7.
num = 0xd8.
num = 0xd9.
num = 0xda.
num = 0xdb.
num = 0xdc.
num = 0xdd.
num = 0xde.
num = 0xdf.
num = 0xdA.
num = 0xdB.
num = 0xdC.
num = 0xdD.
num = 0xdE.
num = 0xdF.
num = 0xE0.
num = 0xE1.
num = 0xE2.
num = 0xE3.
num = 0xE4.
num = 0xE5.
num = 0xE6.
num = 0xE7.
num = 0xE8.
num = 0xE9.
num = 0xEa.
num = 0xEb.
num = 0xEc.
num = 0xEd.
num = 0xEe.
num = 0xEf.
num = 0xEA.
num = 0xEB.
num = 0xEC.
num = 0xED.
num = 0xEE.
num = 0xEF.
```

```
num = 0xe0.
num = 0xe1.
num = 0xe2.
num = 0xe3.
num = 0xe4.
num = 0xe5.
num = 0xe6.
num = 0xe7.
num = 0xe8.
num = 0xe9.
num = 0xea.
num = 0xeb.
num = 0xec.
num = 0xed.
num = 0xee.
num = 0xef.
num = 0xeA.
num = 0xeB.
num = 0xeC.
num = 0xeD.
num = 0xeE.
num = 0xeF.
num = 0xF0.
num = 0xF1.
num = 0xF2.
num = 0xF3.
num = 0xF4.
num = 0xF5.
num = 0xF6.
num = 0xF7.
num = 0xF8.
num = 0xF9.
num = 0xFa.
num = 0xFb.
num = 0xFc.
num = 0xFd.
num = 0xFe.
num = 0xFf.
num = 0xFA.
num = 0xFB.
num = 0xFC.
num = 0xFD.
num = 0xFE.
num = 0xFF.
num = 0xf0.
num = 0xf1.
num = 0xf2.
num = 0xf3.
num = 0xf4.
num = 0xf5.
num = 0xf6.
num = 0xf7.
num = 0xf8.
num = 0xf9.
num = 0xfa.
num = 0xfb.
num = 0xfc.
num = 0xfd.
num = 0xfe.
num = 0xff.
num = 0xfA.
num = 0xfB.
num = 0xfC.
num = 0xfD.
num = 0xfE.
num = 0xfF.
num = 0x000.
num = 0x0Aa.
num = 0xF5B.
num = 0xFFF.
num = 0x0000.
num = 0x0101.
num = 0x4000.
num = 0xCAFE.
num = 0xC0DE.
```

```
num = 0xBABE.
num = 0xBAAD.
num = 0xBEEF.
num = 0xFACE.
num = 0xFEED.
num = 0xFFFF.
num = 0x00000000.
num = 0xcafebabe.
num = 0xcAfEbAbE.
num = 0xCAFEBABE.
num = 0xDEADC0DE.
num = 0xFFFFFFFF.
num = 0x0000000000000000.
num = 0x7FFFFFFFFFFFFFFF.
num = -0x7FFFFFFFFFFFFFFF.
num = -0x8000000000000000.

num = -0x8000000000000001 no-error.
if not error-status:error or not error-status:get-number(1) eq 78 then message "Expected error 78 for assignin
g -0x8000000000000001".

num = 0x8000000000000000 no-error.
if not error-status:error or not error-status:get-number(1) eq 78 then message "Expected error 78 for assignin
g -0x8000000000000001".

num = 0xFEEDFACEBAADBEEF no-error.
if not error-status:error or not error-status:get-number(1) eq 78 then message "Expected error 78 for assignin
g -0x8000000000000001".

num = 0xFFFFFFFFFFFFFFFF no-error.
if not error-status:error or not error-status:get-number(1) eq 78 then message "Expected error 78 for assignin
g -0x8000000000000001".

message "All tests completed successfully.".
```

#3 - 10/11/2017 05:20 PM - Greg Shah

I've created task branch 3353a for changes related to this problem.

#4 - 10/18/2017 01:39 PM - Greg Shah

Revision 11180 of 3353a has a good first pass implementation of hexadecimal literals. It has been tested with full conversion as well as compilation of the converted result.

The support is marked as partial (for both conversion and for runtime). The reason is that where clauses, dynamic queries, schema and DMOs don't yet have support. To find the remaining code, search for NUM_LITERAL references in rules/schema/, rules/annotations/where_clause*, src/com/goldencode/p2j/schema/ and src/com/goldencode/p2j/persist/.

#5 - 10/18/2017 01:39 PM - Greg Shah

- Related to Bug #3352: non-reserved keywords in temp-table field names added

#6 - 10/18/2017 01:39 PM - Greg Shah

- Related to Bug #3354: func_class return type not recognized added

#7 - 10/18/2017 02:08 PM - Greg Shah

- Related to Bug #3351: create widget assign clause doesn't parse the rvalue query qualifier added

#8 - 10/19/2017 11:02 AM - Greg Shah

- Related to Bug #3360: method return type can have an optional EXTENT qualifier added

#9 - 10/19/2017 11:08 AM - Greg Shah

- Status changed from New to Test

- Assignee set to Greg Shah

- % Done changed from 0 to 100

#10 - 10/19/2017 12:00 PM - Greg Shah

- Related to Feature #3343: implement a 4GL syntax extension and the runtime backing to send emails (including attachments) added

#11 - 08/21/2018 01:06 PM - Greg Shah

- Status changed from Test to Closed

The branch 3353a was merged into the trunk in revision 11187. This is now released publicly in FWD v3.2.

FWD v3.2.0 has been released. This issue is being closed accordingly.