# Base Language - Feature #2287

## implement missing type conversion for POLY cases in data type wrapper assign method

04/16/2014 06:47 PM - Eric Faulhaber

| | | | |
|---|---|---|---|
| **Status:** | Closed | **Start date:** | |
| **Priority:** | High | **Due date:** | |
| **Assignee:** | Marius Gligor | **% Done:** | 100% |
| **Category:** | | **Estimated time:** | 80.00 hours |
| **Target version:** | Cleanup and Stablization for Server Features | | |
| **billable:** | No | **vendor_id:** | GCD |

**Description**

**Related issues:**

| | |
|---|---|
| Related to Base Language - Feature #2050: ATTR_POLY/METH_POLY used as rvalue ... | **Closed** |
| Related to Base Language - Feature #2051: conversion support for ATTR_POLY/ME... | **Closed** |

### History

#### #1 - 04/16/2014 06:59 PM - Eric Faulhaber

Several data type wrapper implementations have TODOs in assign(BaseDataType value, boolean force), as in the following example excerpted from the Text class:

```
if (value instanceof Text)
{
   assign((Text) value, force);
}
else
{
   // TODO: implement type conversion here for _POLY cases
}
```

This situation exists in (at least) Text, decimal, handle, and rowid.

Since the else case never performs the assignment, I have had downstream code blow up which relies on the POLY use case in Text (in client's application code) and in decimal (inadvertently in an unrelated test case). The Text case in particular is blocking a customer deliverable, which is why I have set the priority of this issue higher.

#### #2 - 04/17/2014 08:48 AM - Greg Shah

*- Estimated time set to 80.00*

*- Assignee set to Hynek Cihlar*

There are polymorphic functions (FUNC_POLY), attributes (ATTR_POLY) and methods (METH_POLY) in the 4GL. These are built-in language features that have an indeterminate data type at compile time. The type is only resolved at runtime. The most common used of these features are DYNAMIC-FUNCTION and buffer-field-handle:VALUE.

Normally, the 4GL is very sensitive to data type mismatches. For example:

```
if true eq "yes" then message "this can't happen".
```

This will generate the compile failure ** Incompatible data types in expression or assignment. (223).

The exception the 4GL is when one of the _POLY cases is used directly in an expression or assignment. These cases then silently transform the type to the required type (if possible) and will continue execution. In P2J, we handle these cases by providing a constructor that accepts BaseDataType and then we detect the usage of _POLY cases and make sure that the result is wrapped in a constructor like new target-type(DYNAMIC-FUNCTION("some-func")) where target-type is the wrapper type (e.g. character) that is required by the expression. We do analysis on the expression to try to determine this properly. That detection code is not 100% perfect, but it has been made to work for all cases encountered so far.

This example 4GL code shows the use case:

```
def var txt as char init "bogus".

function logical-func returns logical ():
   return true.
end.

if dynamic-function("logical-func") eq "yes" then message "yep".
else message "nope".
```

This will print "yep".

Pretty much all BaseDataType classes have a constructor that is used for this purpose, though it can be in a base class instead of the child classes. These have not yet been implemented. This task is to test the algorithm and limits of the transformations of each type and based on the results, to implement these constructors. For more details on the original conversion implementation/findings, please see #2050 and #2051.

#### #3 - 05/28/2014 12:51 PM - Greg Shah

*- Assignee changed from Hynek Cihlar to Marius Gligor*

#### #4 - 06/02/2014 02:01 AM - Marius Gligor

*- Status changed from New to WIP*

#### #5 - 06/02/2014 12:20 PM - Marius Gligor

*- File mag_upd20140602b.zip added*

1. I've started to work on this issue by executing the provided test case on P4G environment and I found the following conversions to a logical value:

- For character values "yes" = TRUE, "no" = FALSE and for other values = UNKNOWN.
- For numeric values 1 = TRUE, 0 = FALSE and for other values = UNKNOWN
A decimal value for example is converted as follow: 0.0 = FALSE, 1.0 = TRUE, 1.01 = UNKNOWN, 2.0 = UNKNOWN etc.
An integer value for example is converted as follow: 0 = FALSE, 1 = TRUE, 2 = UNKNOWN, ~~1 = UNKNOWN etc.~~
A date value always converted to an UNKNOWN logical value.

2. Then I converted the test case to P2J code and I did a lot of changes to logical wrapper class:
- I changed the code inside the assign(BDT, boolean) method according to my tests on P4G.
- I changed also the code inside compareTo(Object) to compare NumberType and date instances.
- Because logical wrapper could contains UNKNOWN values I added also some code inside compareTo(Object) to fix the cases when unknown values are compared.

```
By doing those changes inside the logical class the P2J converted test case works like the P4G test.
```

#### #6 - 06/02/2014 02:11 PM - Greg Shah

Code Review 0602b

The changes look good.  I assume you are working on the same changes for the other BDT types.

#### #7 - 06/03/2014 07:17 AM - Marius Gligor

1. Working to add more BDT types for conversion inside the logical class and I found a blocked issue.
I have the following P4G use case:

```
def var dec-val as decimal init 45.

message logical(dec-val).
message decimal(true).

function logical-func returns logical ():
    return true.
end.

if dynamic-function("logical-func") eq dec-val
    then message "yep"
    else message "nope".
```

When I executed the test

message logical(dec-val).  display "yes"
and
message decimal(true). display 1

which means that
- a numeric value 0 is converted to false and any other values except unknown are converted to true.
- a logical true value is converted to 1 and a false value is converted to 0.

However executing the statement:  if dynamic-function("logical-func") eq dec-val
a "nope" string is displayed which means that on P4G the result of the  dynamic-function("logical-func") is evaluated and converted to a decimal value which is 1.
Comparing 1 with 45 results in a "nope" message. The dec-val type is known at compile time and P4G try to convert the evaluated type to this known type.

The P2J converted code for this test case looks like:

```
/**
 * Business logic (converted to Java from the 4GL source code
 * in poly-1.p).
 */
public class Poly1
{
   /**
    * External procedure (converted to Java from the 4GL source code
    * in poly-1.p).
    */
   public void execute()
   {
      externalProcedure(new Block()
      {
         decimal decVal = new decimal(45);

         public void init()
         {
            TransactionManager.registerUndo(decVal);
         }

         public void body()
         {
            message(new logical(decVal));
            message(new decimal(new logical(true)));
            if (_isEqual(ControlFlowOps.invokeDynamicFunction(logical.class, "logical-func"), decVal))
            {
               message("yep");
            }
            else
            {
               message("nope");
            }
         }
      });
   }

   public logical logicalFunc()
   {
      return function(logical.class, new Block()
      {
         public void body()
         {
            returnNormal(true);
         }
      });
   }
}
```

When this expression: _isEqual(ControlFlowOps.invokeDynamicFunction(logical.class, "logical-func"), decVal)
is evaluated in P2J the second parameter type is converted to first parameter type which is different than P4G evaluation!

Also in P2J instead to have a value of 1 for message(new decimal(new logical(true))); a value of 0.0000000001 (1E-10) is displayed because on the decimal class the following value is defined for a logical value of true.

```
/** Constant for BigDecimal value 1 with scale MAX_SCALE */
   static final BigDecimal ONE = new BigDecimal(BigInteger.ONE, MAX_SCALE);
```

I think that this is a bug that should be fixed.

2. Another problem that I found is related to initial values for P4G primitives.
According to my tests decimal, int64, integer types are initialized with 0 if no initial values are specified in P4G.
In P2J the default constructors for these types initialize the instances with an unknown value.
Logical values are initialized to false in P4G and to unknown value in P2J.
Character values are initialized to an empty string in P4G and to unknown value in P2J.

#### #8 - 06/03/2014 07:23 AM - Constantin Asofiei

Marius Gligor wrote:

> 2. Another problem that I found is related to initial values for P4G primitives.

I think this might be a V9/V10 difference: how is your p2j.cfg.xml configured? You need to have this:

```
      <parameter name="source-code-version" value="10.2B"/>
```

to activate 10.2B compatibility during conversion.

#### #9 - 06/03/2014 08:05 AM - Marius Gligor

I have no such line in my p2j.cfg.xml.
My configuration file is for <!-- Majic-specific P2J main configuration -->

#### #10 - 06/03/2014 08:09 AM - Constantin Asofiei

Marius Gligor wrote:

> I have no such line in my p2j.cfg.xml.
> My configuration file is for <!-- Majic-specific P2J main configuration -->

OK, then please add it. Note that MAJIC is built in v9 and we can't break this compatibility. That's why source-code-version needs to be set to 10.2b, when checking stuff against windev01/lindev01.

#### #11 - 06/03/2014 09:09 AM - Marius Gligor

I added the line in my configuration file and now works properly.

Thanks

#### #12 - 06/03/2014 09:30 AM - Marius Gligor

*- File mag_upd20140603a.zip added*

I added new BDT types for conversion to a logical value.
I found and implemented the conversion rules for date and handle types.
I changed the conversion rule for NumericType values to works like on P4GL
The conversion now works like on P4GL logical(expression).

#### #13 - 06/03/2014 09:55 AM - Greg Shah

I haven't reviewed your changes yet. But while you were editing the code, I wrote this:

> When this expression: _isEqual(ControlFlowOps.invokeDynamicFunction(logical.class, "logical-func"), decVal) is evaluated in P2J the second parameter type is converted to first parameter type which is different than P4G evaluation!

This means that the left-side operand needs to be wrapped in a decimal constructor like this:

```
_isEqual(new decimal(ControlFlowOps.invokeDynamicFunction(logical.class, "logical-func")), decVal)
```

I think that would solve the problem. This means that we have some conversion changes that are needed to implement additional wrapping. To see the TRPL rules that are involved in wrapping today, you can search for kw_dyn_func in the following rule-sets:

rules/annotations/functions.rules
rules/annotations/deferred_logical_evaluation.rules
rules/convert/base_structure.xml
rules/convert/builtin_functions.rules
rules/convert/literals.rules

These rules all "communicate" using some annotations ("ignorecast", "classname", "casttype"...). Earlier rule-sets calculate the annotations (these will call something like "putAnnotation" or "putNote") and leave them behind to direct the downstream rule-sets' behavior (which will call something like "isNote" or "getNoteString" or "getAnnotation").

I think the annotation that you will care about is "chp_wrapper". Search through base_structure.xml and builtin_functions.rules to see how that is used.

Finally, it is important for you to understand how we calculate the type of expressions and sub-expressions (every sub-tree of an expression has a type and the data/operators can be analyzed to determine that type statically). We sometimes know implicitly what the type will be by context, but generally we make calls to ExpressionConversionWorker.expressionType() to do the calculation. When we are faced with the DYNAMIC-FUNCTION cases (and also some cases like the BUFFER-VALUE attribute), the type is truly polymorphic. However, for the DYNAMIC-FUNCTION cases, we use some "tricks" to see if in fact the type is static. The primary trick used is when the function name in a DYNAMIC-FUNCTION invocation is a string literal, we lookup that function name in a map of all user-defined functions and if there is only 1 return type associated with that function name (in the whole project), then that must be the type of the return value for that DYNAMIC-FUNCTION call. This trick almost always works because, in practice, the use of DYNAMIC-FUNCTION is usually hard coded with a string literal function name AND there is usually only 1 definition of that in the project (or the same definition is duplicated in many places, which is essentially the same thing).

The other thing to know is that if we can't figure out the return type by this static analysis, we fall back to analyzing the context around call. That usually means we must look UP the tree to see how the return value is being used. You can see this in the ExpressionConversionWorker.analyzeContext() code.

I suspect that we must add some special cases where we need to do extra wrapping of the return value to duplicate the proper 4GL results.

```
/** Constant for BigDecimal value 1 with scale MAX_SCALE */
static final BigDecimal ONE = new BigDecimal(BigInteger.ONE, MAX_SCALE);
```

I think that this is a bug that should be fixed.

Agreed.  Please fix it.

#### #14 - 06/03/2014 10:05 AM - Greg Shah

Code Review 0603a

The changes look good.  I'm glad you were able to fix the problem without extra conversion-time wrapping.

#### #15 - 06/03/2014 11:29 AM - Marius Gligor

Until to fix the conversion issue I found a temporary solution to test the behaviour on P2J by swap the _isEqual operands :-)

```
if (_isEqual(decVal, ControlFlowOps.invokeDynamicFunction(logical.class, "logical-func")))
```

#### #16 - 06/03/2014 12:34 PM - Marius Gligor

*- File mag_upd20140603b.zip added*

The archive contains the implementation of decimal wrapper type conversion for POLY cases.

#### #17 - 06/03/2014 02:41 PM - Greg Shah

Code Review 0603b

The changes look good.

#### #18 - 06/04/2014 09:56 AM - Constantin Asofiei

Greg: will the DynamicOps APIs be implemented with this task or another?

#### #19 - 06/04/2014 10:02 AM - Greg Shah

Excellent point/question.

Marius: please do implement the missing functionality in the util/DynamicOps class as part of this work.  See the TODOs in the plus() and minus() methods.

#### #20 - 06/06/2014 11:00 AM - Marius Gligor

*- File mag_upd20140606a.zip added*

Here are the latest changes that I did so far for other BDT's like handle, longchar, raw, memptr, date, raw etc.

In order to know if a BDT is the result of a DYNAMIC-FUNCTION call I have an idea.

In the abstract BaseDataType class we have to add a boolean field let say "dynamic = false" which indicate if the value is a static or a dynamic created (default static).

When ControlFlowOps.invokeDynamicFunction is executed a BDT instance is returned having the indication that is the result of a dynamic call. The last step in execution is the call of ControlFlowOps.invokeImpl and here we could set the "dynamic = true" flag inside the BDT before return. This flag could help us in further evaluations. What do you think about this?

#### #21 - 06/06/2014 11:20 AM - Constantin Asofiei

Marius Gligor wrote:

> In order to know if a BDT is the result of a DYNAMIC-FUNCTION call I have an idea.

Your idea looks good, but I'd like first to see why you need it: isn't calling assign(BDT) enough to assume we are in a _POLY case?

More, keep in mind that beside DYNAMIC-FUNCTION there are BUFFER-VALUE and the :: operator.

#### #22 - 06/06/2014 11:21 AM - Marius Gligor

regarding "dynamic" flag I think that is better to set on this call ControlFlowOps.invokeDynamicFunctionImpl

#### #23 - 06/06/2014 12:28 PM - Greg Shah

I would really like to avoid such a flag. The reason is simple: I want the wrapper classes (BDT subclasses) to limit their "knowledge" and dependencies upon the features of the 4GL that are not directly related to the data types. In this case, the concept of dynamic vs static type compatibility is a bit further off from the core data type roles than I would prefer. We try to keep these data types "lean" because they are used outside of P2J (e.g. by Java code that connects remotely to P2J servers). Being small and lean is also good because these are constantly serialized when we transfer to the client.

That is why we have tried to implement the BDT constructor approach when needed and otherwise the wrappers themselves don't have to carry the burden of this obscure feature.

#### #24 - 06/09/2014 12:02 PM - Marius Gligor

For DynamicOps.plus() and DynamicOps.minus() implementation I'm using the following test case:

```
def var v1 as deci.
def var v2 as int init 45.
def var v3 as handle.
def var v4 as handle.
def var v5 as int64 init 67.
def var v6 as char.
def var v7 as longchar.
def var v8 as date init 1/1/1.
def var v9 as datetime init now.

def button b1.
def button b2.
def frame a b1 b2.
```

```
v3 = b1:handle.
v4 = b2:handle.

function f2 returns date ():
    return v8.
end.

message v1 + dynamic-function("f2").
message v1 - dynamic-function("f2").
message dynamic-function("f2") - v1.
```

and I changed the return type for function "f2" and the v1 types.
The type of evaluated expression depends on operand types and seems to have some priority order: longchar, character, date, double, int64 and integer.
The tests reveals some interesting cases related to the type of the evaluated expression:

1. for plus()
- if at least one operand op1 or op2 is of type longchar the result is longchar.
- if at least one operand op1 or op2 is of type character the result is character.
- if at least one operand op1 or op2 is of type date the results is date. However if both op1 and op2 are of type date error 223 is throw (Incompatible data types in expression or assignment.) Also datetime and datetimetz does not works at all (the same 223 error).
- if at least one operand op1 or op2 is of type decimal the results is decimal.
- if at least one operand op1 or op2 is of type int64 the results is int64.
- if at least one operand op1 or op2 is of type integer the results is integer.

Other types like HANDLE and LOGICAL works only if the other operand is of type character or longchar and the result is of type character or longchar.

2. for minus()
- longchar, character, datetime, datetimetz types are not allowed at all (error 223).
- if op1 is date and op2 is date the rsult is integer (decimal)!!!
- if op1 is date and op2 is number (int, int64, decimal) the result op1 - op2 is date.
- if op2 is date and op1 is not date or decimal error 223 is throw.
- if at least one operand op1 or op2 is of type decimal the results is decimal.
- if at least one operand op1 or op2 is of type int64 the results is int64.
- if at least one operand op1 or op2 is of type integer the results is integer.

Other types like HANDLE and LOGICAL works only if the other operand is of type decimal and the result is decimal.

MEMPTR and RAW seems to be not supported (error 223).

#### #25 - 06/09/2014 12:16 PM - Constantin Asofiei

Marius Gligor wrote:

> message v1 + dynamic-function("f2").
> message v1 - dynamic-function("f2").
> message dynamic-function("f2") - v1.

Only a quick note: message can be seen as a kind of "poly" l-value, as when it evaluates the expression I think it doesn't pose any constraints/casting on the result. But what happens if the l-value is a variable or a real _POLY, like the :: operator, i.e.

```
def var i as int.
def temp-table tt1 field f1 as int.
create tt1.
tt1::f1 = <some-poly-expression>.
i = <some-poly-expression>.
```

Is the behavior the same as in your message cases?

#### #26 - 06/10/2014 11:35 AM - Marius Gligor

*- File mag_upd20140610a.zip added*

Today I did tests using test cases like Constantin suggested. The results are the same as for message cases.
The expression (rvalue) is evaluated based on operator operands only.
When the rvalue is assigned to a lvalue the rvalue is converted to lvalue type.
According to my tests results I implemented the plus and minus operators on DynamicOps class.

#### #27 - 06/10/2014 01:45 PM - Greg Shah

Code Review 0610a

It looks very good. Some notes:

1. For the raw(BDT) constructor, please check if a properly encoded RAW instance (using raw.toStringMessage()) will be decoded properly by the 4GL.

2. Why do you have FIXME notices in the DynamicOps.java? 4GL Restrictions are not things to fix or change. If you have the behavior matching that of the 4GL, then it is just right and needs no TODO or FIXME. It is always useful to leave behind a notice that those restrictions are matching the 4GL.

3. In date.compareTo(), the else if (!(obj instanceof date)) can just be an else since a date instance will follow the if (obj instanceof BaseDataType) branch.

4. Is there anything more to do for this task?

#### #28 - 06/11/2014 08:24 AM - Marius Gligor

*- File mag_upd20140611a.zip added*

1. I've fixed the issues from your code review.

2. Regarding the RAW type on the customer's Linux server I've got the following error message when I executed the script:

```
DEFINE VAR r1 AS RAW.

FIND FIRST cust.
r1 = RAW(name).



PROGRESS dataserver does not support RAW access to fields. (1859)
```

On the customer's Windows server I've got the following error when I tried to execute the same script via a remote desktop connection on both ChUI and GUI Progress environment.

```
The licence for the client executable has expired (4399).
```

However I changed the decimal and logical classes to avoid conversion from RAW, MEMPTR or ROWID types.

3. On this issue remains to fix the wrapping of dynamic invocation result type (see #13)
For DynamicOps.plus and DynamicOps.minus we have to do no wrappings but for operators like eq, lt, gt, le, ge etc. we have to fix the P2J code to works like on P4GL.

#### #29 - 06/11/2014 10:02 AM - Greg Shah

Code Review 0611a

The changes look good.

> 2. Regarding the RAW type on the customer's Linux server I've got the following error message when I executed the script:

It is important to avoid database usage unless you are really trying to work on database issues. In your example, you are referencing a database field when you could be using a variable. Please make sure to test the variable cases for the same types, since the database behavior differences may be obscuring the results.

If you are using the p2j_test schema, I'm not sure what the "name" field is that you are referencing since the "cust" table has no "name" field. So in your example, it is not clear what this error really means.

> PROGRESS dataserver does not support RAW access to fields. (1859)

I'm not sure what this means. RAW is a valid datatype for database fields (although MEMPTR is not).

> The licence for the client executable has expired (4399).

Strange. I am able to use the prowin32 environment on windev01 without issue (but I wasn't running that specific script).

> 3. On this issue remains to fix the wrapping of dynamic invocation result type (see #13)
> For DynamicOps.plus and DynamicOps.minus we have to do no wrappings but for operators like eq, lt, gt, le, ge etc. we have to fix the P2J code to works like on P4GL.

OK. Please try to limit the changes just to the dynamic cases because for the vast majority of usage, these operators don't need the special _POLY behavior. In fact, I think that static usage of the same datatypes will fail where dynamic usage may succeed, so we need to match that behavior.

#### #30 - 06/12/2014 06:15 AM - Marius Gligor

*- File mag_upd20140612a.zip added*

Unfortunately my idea to fix the note #13 does not work. Using the following test case:

```
def var v1 as integer init -43.
def var v2 as logical init true.

function f1 returns logical ():
    return true.
end.

function f2 returns integer ():
    return -43.
end.

if dynamic-function("f1") eq v1 then message "yep". else message "nope".
if v1 eq dynamic-function("f1") then message "yep". else message "nope".

if dynamic-function("f2") eq v2 then message "yep". else message "nope".
if v2 eq dynamic-function("f2") then message "yep". else message "nope".
```

I've got:

nope
nope
yep
yep

which prove once again that in P4GL the result of dynamic-function invocation is converted to the known data type of the other operand.

This issue is specific only if one of the operand is of type logical due to the specific conversions of this type:
If v1 is integer and the result of f1 is converted to integer we have v1=-43 and f1=1 which means "nope"
If v2 is logical and the result of f2 is converted to logical we have v2=true and f1=true which means "yep"

In other words converting integer to logical and than back to integer does not produce always the same values.
Converting integer v1 -> logical v2 -> integer v3 we have v1 = v3 only if v1 is 0 or 1.

For other BDT types except logical seems to work because the conversion between types is comparable regarding the static or dynamic nature.

In P2J the method compareTO is used to implements the evaluation of "eq" expression.
I tried to fix this issue inside the compareTo method of logical type but unfortunately it's doesn't work on all cases.

The only solution remains to change the conversion and to wrap the result of a dynamic-function call to the type of the other operand if this is a static one.

However handle is a special type which for example does not allow constructor with other BDT types but it's comparable and can be used in expressions.
The implementation of comapreTO method for handle is not polymorphic (no conversions from BDT types are made).

#### #31 - 06/12/2014 09:07 AM - Greg Shah

Code Review 0612a

The changes look fine.

> The only solution remains to change the conversion and to wrap the result of a dynamic-function call to the type of the other operand if this is a static one.

Understood.

#### #32 - 06/13/2014 09:45 AM - Marius Gligor

Here are the latest changes that I made on this issue.

1. I implemented a rule to wrap the result of a DYNAMIC-FUNCTION call on expression like EQUALS, NOT_EQ, GT, LT, GTE, LTE into a BTD instance.
The rule define also some restrictions:
- the second parameter of the expression must be defined static and must be a BDT type.
- The wrapper is generated only if at least one of the parameter is of type logical.
We could extend the wrapping rule to other types also except the handle type due to its implementation in P2J.
For other types the code in compareTo method is enough to implement the evaluation of expression because conversions for other types are idempotent.
For logical type the conversion is not idempotent (converting a logical value to integer and back to logical produce different results except for 0 and 1) and we must use the wrapper in order to have the same behaviour like on P4GL.

2. The issue #2050 seems to be tightly related to this one and I think that we could investigate also the #2050 issue during the work on this issue.

#### #33 - 06/13/2014 09:45 AM - Marius Gligor

*- File mag_upd20140613a.zip added*

Here is the attachement.

#### #34 - 06/13/2014 10:53 AM - Greg Shah

Code Review 0613a

1. The datetimetz imports have been expanded.

2. In functions.rules, this code is too restrictive:

```
         <rule>upPath("EXPRESSION/EQUALS") or upPath("EXPRESSION/NOT_EQ") or
               upPath("EXPRESSION/GT")     or upPath("EXPRESSION/LT")     or
               upPath("EXPRESSION/GTE")    or upPath("EXPRESSION/LTE")
```

There are many different possible parent nodes for these operators.  The above code will work for some "top-level" usage of these operators, but there are missing cases.

Why not just look for the operator nodes directly (parent.type == prog.equals)?

3. Your changes to builtin_functions.rules seem like they will break other usage of the chp_wrapper "facility".  I think it is used for more than just binary operator parent nodes, so this is probably not correct.

4. About this:

> 2. The issue #2050 seems to be tightly related to this one and I think that we could investigate also the #2050 issue during the work on this issue.

I completely agree.

**#35 - 06/13/2014 11:40 AM - Marius Gligor**

This is my first issue on which I'm working and require to define new rules.
I have one question please regarding rules.

Inside builtin_functions.rules I found:

<rule>
....
</rule>

<rule on="false">
....
</rule>

According to my tests the second rule <rule on="false"> is never executed!

So I did some changes to looks like:

<rule>
....
<rule on="false">
....
</rule>
</rule>

And now it works. Is this OK or not?

#### #36 - 06/13/2014 12:02 PM - Marius Gligor

OK. Now I understand. The real structure was:

```
<rule>
   <rule>
   ....
   </rule>

   <rule on="false">
   ....
   </rule>
</rule>
```

and rule on="false" is related to the top rule.

#### #37 - 06/13/2014 12:33 PM - Marius Gligor

*- File mag_upd20140613b.zip added*

I did the changes in the rules. Please let me know if my changes are OK.

#### #38 - 06/13/2014 12:34 PM - Greg Shah

Exactly right.  The on="false" is a way to make the contained rule part of the else branch of the conditional.

#### #39 - 06/13/2014 01:51 PM - Greg Shah

Code Review 0613b

This code in builtin_funcs.rules is definitely not right:

```
               <rule>ftype == prog.func_poly
                  <!-- wrap this function call in another type -->
                  <rule on="false">isNote("chp_wrapper")
                     <action>cls = getNoteString("chp_wrapper")</action>
                     <!-- function return type if any -->
                     <action>casttype = this.getAnnotation("casttype")</action>
                     <rule>cls != null and cls != casttype and (casttype == "logical" or cls == "logical")
                        <action>
                           lastid = createPeerAst(java.constructor,
                                                  cls,
                                                  closestPeerId)
                        </action>
                     </rule>
                  </rule>
               </rule>
```

It will always trigger since ftype should never be func_poly.  So this will trigger whenever there is a chp_wrapper annotation that has a classname that is not the same as casttype and casttype is logical.

This definitely should only be triggered in the kw_dyn_func case.  The previous code (0613a) definitely changed the else logic of the kw_dyn_func

section of rules in a way that would break it.  If you use chp_wrapper, that code will need to be placed in the if portion of the kw_dyn_func section.

It is also a good idea to avoid adding the chp_wrapper annotation (in annotations/functions.rules) except in the exact case that you need (the classname is not logical and the other operand of a binary operator is static and is logical).  That will simplify the builtin_functions.rules code and make it less likely that this annotation will have unintended consequences later.

#### #40 - 06/13/2014 02:02 PM - Marius Gligor

When my rule from annotations/functions.rules is executed the return type of the function is not yet known. I think that is fixed later on another rule. I have to recheck.
Do you think that is a good idea to use another annotation instead of "chp_wrapper" for this case?

#### #41 - 06/13/2014 02:05 PM - Greg Shah

> Do you think that is a good idea to use another annotation instead of "chp_wrapper" for this case?

Yes, it will make things clearer and safer.

Perhaps it will also help because you can locate your annotations logic at a later point in the process, when enough information is available.

#### #42 - 06/16/2014 11:38 AM - Marius Gligor

*- File mag_upd20140616a.zip added*

Here are the new changes related to this issue and #2050.

1. I created a new annotation "poly_wrapper" to be used on this case.
I moved the rule inside file annotations/functions.rule in a position where we can get also the function result type.
The rule form convert/builtin_functions.rule is now a simple rule.

2. The solution for #2050 is to wrap (cast) the return result of DYNAMIC-FUNCTION to the known type as P4GL do.
When the test case from note 24 is converted in P2J the AST node type is of type DATE_LITERAL, DATETIME_LITERAL, DATETIME_TZ_LITERAL or FUNC_POLY.
For DATE_LITERAL, DATETIME_LITERAL, DATETIME_TZ_LITERAL I created a new rule to warp to date, datetime and datetimetz.
With the cast in place the results are the same like on P4GL.

3. If both operands are of type FUNC_POLY no wrapping is generated.

```
- If the types of return functions are the same it works like on P4GL because compareTo() using same types alr
eady return 0 (equals)
```

```
- In P4GL the result of evaluation is unknown (?) if the types of return functions are different.
     In P2J the compareTo() is used to evaluate expression which results in a different value than unknown.
     Should we force always and unknown result in such cases or it is a special case which should be avoided a
t conversion time?
     How we could force a unknown value on this case?
```

4. I extended the wrapping rule to be generated also on other cases for static values due to my conclusions from #2050 castings.

#### #43 - 06/16/2014 12:42 PM - Constantin Asofiei

Marius, when regression testing this update, make sure to check the converted sources for the server project, if there are any differences.

#### #44 - 06/16/2014 01:34 PM - Marius Gligor

Constantin, please could you offer me more details? What is server project, where is located, how to do the test?

#### #45 - 06/16/2014 03:23 PM - Greg Shah

Code Review 0616a

The approach is much better.  It is almost done now.  The final issue is this code (from annotations/functions.rules):

```
        <!-- wrap return type if static BDT -->
                  <rule>ref.isAnnotation("refid")
                     <action>refid = ref.getAnnotation("refid")</action>
                     <action>ref = getAst(refid)</action>
                     <rule>ref != null and ref.isAnnotation("classname")
                        <action>cls = ref.getAnnotation("classname")</action>
                        <!-- cls should be a BDT type like -->
                        <rule>cls == "character"  or cls == "longchar" or
                              cls == "decimal"    or cls == "int64"    or
                              cls == "integer"    or cls == "recid"    or
                              cls == "date"       or cls == "datetime" or
                              cls == "datetimetz" or cls == "logical"
                           <rule>casttype == null or (casttype != cls and casttype != "handle")
                              <action>putNote("poly_wrapper", cls)</action>
                           </rule>
                        </rule>
                     </rule>
                  </rule>
```

The problem with this code is that it only works when the other operand is a variable.  The other operand could be any sub-expression including a user-defined function, a handle based method/attribute, a literal, the result of an operator...

We can't hard code all these different cases here, just like handling all possible cases everywhere else would be prohibitive.  Instead, we centralize and delegate the type-analysis of sub-expressions to the ExpressionConversionWorker.  In annotations/functions.rules, you can see calls to ecw.expressionType().  This calls the ExpressionConversionWorker and let's it detect the type of the given node.  Please use this technique instead of the refid/classname processing approach.  The result will still be a string that has the wrapper type name of the node if it can be detected.

The only concern I have is that if the operand has a non-static type (it is also a _POLY), will the the ExpressionConversionWorker approach handle this properly.  It should, but I'm don't know for sure, so keep an eye on that.

#### #46 - 06/16/2014 03:29 PM - Greg Shah

In regard to how to setup and run conversion on the server project, please see #2299.  Read that task history carefully and all the way through.  You can do that setup on your local system IF and ONLY IF your workstation is installed/configured according to the Golden Code standard workstation documentation/scripts.  You may also do this on devsrv01.  Make sure the project is setup inside a ~/<customer_name>/ directory so that our antivirus runs don't scan that dir tree daily.

You only have to get the project setup and converting.  You don't need to do anything for database setup/import or for getting a server running.

You should run conversion on this project using the current bzr revision of P2J.  Save off the generated results for comparison later.  Then apply your changes and run conversion again.  Then compare the generated results of the latest conversion run, with your saved results.  Any changes that are not expected are probably bugs.

Finally, make sure the converted code compiles and jars.

LE: GES removed the customer name from this entry.

#### #47 - 06/17/2014 03:53 AM - Marius Gligor

*- File mag_upd20140617a.zip added*

Using ExpressionConversionWorker the rule in annotations/functions.rules became more simple and cover also the case of DATE_LITERAL, DATETIME_LITERAL and DATETIME_TZ_LITERAL.

#### #48 - 06/17/2014 07:20 AM - Greg Shah

Code Review 0617a

The changes are good.

Is there anything else to do for this task (or for #2050)?  If not, then please run both conversion and runtime regression testing.

#### #49 - 06/17/2014 07:30 AM - Marius Gligor

I started the customer server conversion on my workstation as is described on #2299.
For P2J I used the current development repository: bzr checkout ~/repo/p2j_repo/p2j
Should I use another repository for P2J or it is OK to use the ~/repo/p2j_repo repository?

#### #50 - 06/17/2014 08:08 AM - Greg Shah

There is only one P2J repo, so you are OK.

#### #51 - 06/17/2014 12:50 PM - Marius Gligor

I started the customer's server conversion on my workstation. The conversion took 4 hours 58 minutes and 41 seconds.
In the end the build of server project fails due to the following error:
/home/mag/synlib01_NAS/convert/2010A08_06/server/build.xml:393: jar doesn't support the "zip64Mode" attribute

Inside build.xml I found zip4Mode="always" which require ant 1.9.1 or greater.
Unfortunately my ant version is 1.8.2 that's why I've got the error.
Is the server project build the last step in conversion?
If yes I have to upgrade my ant builder and retry the build.

```
.../vcpl/voidv1.p.jast
.../vcpl/voidv2.p.jast
.../vcpl/voidv3.p.jast
Elapsed job time:  00:13:49.587

-------------------------------------------------------------------------------
Elapsed job time:  04:58:41

real    298m41.932s
user    254m20.852s
sys     6m43.080s
Buildfile: /home/mag/synlib01_NAS/convert/2010A08_06/server/build.xml

dir.check:

check.p2j.dir:

p2j-compile:

prepare:

chk_antlr_progress:

antlr_progress:

chk_antlr_expression_evaluator:

antlr_expression_evaluator:

chk_antlr_text:

antlr_text:

chk_antlr_braces:

antlr_braces:

chk_antlr_e4gl:

antlr_e4gl:

chk_antlr_schema:

antlr_schema:

chk_antlr_expression_compiler:

antlr_expression_compiler:

chk_antlr_hql:

antlr_hql:

antlr:

compile:

p2j-jar:

prepare:

chk_antlr_progress:

antlr_progress:

chk_antlr_expression_evaluator:

antlr_expression_evaluator:

chk_antlr_text:

antlr_text:

chk_antlr_braces:
```

```
antlr_braces:

chk_antlr_e4gl:

antlr_e4gl:

chk_antlr_schema:

antlr_schema:

chk_antlr_expression_compiler:

antlr_expression_compiler:

chk_antlr_hql:

antlr_hql:

antlr:

compile:

jar:

prepare:
    [mkdir] Created dir: /home/mag/synlib01_NAS/convert/2010A08_06/server/build
    [mkdir] Created dir: /home/mag/synlib01_NAS/convert/2010A08_06/server/build/lib
    [mkdir] Created dir: /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [mkdir] Created dir: /home/mag/synlib01_NAS/convert/2010A08_06/server/distribution
    [mkdir] Created dir: /home/mag/synlib01_NAS/convert/2010A08_06/server/distribution/docs/api
     [copy] Copying 1 file to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [copy] Copying 9221 files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [copy] Copying 1 file to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [copy] Copying 4 files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes/data
     [copy] Copying 2 files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [copy] Copying 6 files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [copy] Copying 4 files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes/cfg

compile:
    [javac] Compiling 18434 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
    [javac] Compiling 11545 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [javac] Compiling 6007 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [javac] Compiling 2326 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [javac] Compiling 2034 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
    [javac] Compiling 8649 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
     [echo] Compiling Java Sources Done.

jar:

BUILD FAILED
/home/mag/synlib01_NAS/convert/2010A08_06/server/build.xml:393: jar doesn't support the "zip64Mode" attribute

Total time: 28 minutes 0 seconds
mag@mag:~/synlib01_NAS/convert/2010A08_06/server$
```

LE: GES removed a customer app name from some pathing in this entry.

#### #52 - 06/17/2014 01:32 PM - Greg Shah

For now, please change zip64Mode="always" to zip64Mode="as-needed". This change will be checked in soon, but for now it will be needed to build on your system.

> Is the server project build the last step in conversion?
> If yes I have to upgrade my ant builder and retry the build.

I think so, but you'll have to check.

#### #53 - 06/18/2014 03:24 PM - Marius Gligor

On the customer's Server conversion I saved the initial succeeded conversion.
Today I applied my changes on P2J project, I did a new checkout for server project and I prepared the project for conversion.
The conversion took 6 hours. Unfortunately the converted code was not compiled:

```
compile:
[javac] Compiling 18434 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.
[javac] Compiling 11545 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] Compiling 6007 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] /home/mag/synlib01_NAS/convert/2010A08_06/server/src/com/something/server/common/Apenndo0.java:4695: error: no suitable method found for toUpperCase(integer)
[javac]               toUpperCase(new integer(ControlFlowOps.invokeDynamicFunctionIn(character.class, "getForeignValues", targetProcedure())))
[javac]               ^
[javac]     method TextOps.<T>toUpperCase(T) is not applicable
[javac]       (inferred type does not conform to declared bound(s)
[javac]         inferred: integer
[javac]         bound(s): Text)
[javac]     method TextOps.toUpperCase(String) is not applicable
[javac]       (actual argument integer cannot be converted to String by method invocation conversion)
[javac]     method TextOps.toUpperCase(String,boolean) is not applicable
[javac]       (actual and formal argument lists differ in length)
[javac]   where T is a type-variable:
[javac]     T extends Text declared in method <T>toUpperCase(T)
[javac] 1 error
```

The cast new integer() comes from my new rule. So I have to check why this happen and make the appropriate changes in rule.
I think that I have to repeat the conversion test after correction.

#### #54 - 06/18/2014 03:26 PM - Constantin Asofiei

Marius Gligor wrote:

> Unfortunately the converted code was not compiled:

It's not just a matter for the code to compile, you need to diff the converted sources (with and without your update) and check if there are any other differences.

#### #55 - 06/18/2014 03:40 PM - Marius Gligor

Yes I know that and will do. In the mean time I changed the Apenndo0.java class by hand and I did a rebuild of the project (ant jar).
The project is now build successful.

compile:
[javac] Compiling 896 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] Compiling 2326 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] Compiling 2034 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[javac] Compiling 8649 source files to /home/mag/synlib01_NAS/convert/2010A08_06/server/build/classes
[echo] Compiling Java Sources Done.

jar:
[jar] Building jar: /home/mag/synlib01_NAS/convert/2010A08_06/server/build/lib/<app_name>_dmo.jar
[jar] Building jar: /home/mag/synlib01_NAS/convert/2010A08_06/server/build/lib/<app_name>_server.jar

BUILD SUCCESSFUL
Total time: 21 minutes 12 seconds

LE: GES removed a customer's app name from this entry.

#### #56 - 06/18/2014 04:12 PM - Marius Gligor

*- File diff.txt added*

Here the other differences in generated Java code. The first two are OK because we have a NOT-EQ expression.
The third difference in Cttrddo0.java seems to be the same like in Apenndo0.java but fortunately is a compiled statement.
Both differences in Cttrddo0.java and in Apenndo0.java are strange because my rule condition is that the parent is an expression like EQUALS, NOT_EQ, GT, LT, GTE, LTE.
I have to check why the rule is applied to these nodes.

#### #57 - 06/19/2014 03:19 AM - Marius Gligor

Here is the analyse of unexpected differences from the customer's server conversion. The affected generated files are Apenndo0.java and Cttrddo0.java.
Bellow I extracted the P4G snipped code and the corresponding Java converted code affected by my changes,
The cause of differences is the WHERE clause. A construction like:

WHERE a = b

is interpreted as a EQUALS expression and if one of the parameter is a dynamic function the casting rule is applied.

For apenndo0.h script in line 352 an explicit cast is made and the Java cast in line 4067 is the result of converting of the P4G INTEGER statement and is not generated by my casting rule. The convesion for this statement is the same in the initial customer server conversion.
The WHERE clause from line 745 is converted as integer by applying my new casting rule but the problem here is the Java toUpperCase() function which require a String value not an integer value.
As you can see no such conversion exists in the original P4GL code! The toUpperCase() is generated by another rule.
Without inserting toUpperCase the generated code is fully compiled.

For cttrddo0.w the cast is generated by statement:
WHERE cop.cteam-cde = {fn FcTeamCde LhContainerSource}

and the generated Java code is fully compiled.

The generated cast on Cinvddo0.java and Cpijcdo0.java are expected because they are _isNotEqual(op1, op2) expressions.
I have to check if the casting rule is applied also on WHERE expressions in P4GL.
If yes the rule which generate toUpperCase() should be changed if not the casting rule should be changed to avoid casting for WHERE expressions.

```
apenndo0.h

line: 352
    FIND apiEntity
        WHERE apiEntity.apiEntity_ID = INTEGER({fn getForeignValues})
        EXCLUSIVE-LOCK.

line: 4067
      new FindQuery(apientity, "apientity.apientityId = ?", null, "apientity.apientityId asc", new Object[]
      {
         new integer(ControlFlowOps.invokeDynamicFunctionIn(character.class, "getForeignValues", targetProcedu
re()))
      }, LockType.EXCLUSIVE).unique();

line: 745
   FIND apiEntity
      WHERE apiEntity.apiEntity_Id = {fn getForeignValues}
      NO-LOCK.

Apenndo0.java

line: 4963
      new FindQuery(apientity, "apientity.apientityId = ?", null, "apientity.apientityId asc", new Object[]
      {
         toUpperCase(new integer(ControlFlowOps.invokeDynamicFunctionIn(character.class, "getForeignValues", t
argetProcedure())))
      }, LockType.NONE).unique();

    END.

cttrddo0.w
line: 765
      FOR EACH cop WHERE cop.cteam-cde = {fn FcTeamCde LhContainerSource} NO-LOCK,
       EACH cOpSorTrd WHERE cOpSorTrd.cOp_Id = cop.cOp_Id
            AND (copSOrTrd.EndDat = ? OR copSOrTrd.EndDat >= TODAY) NO-LOCK:
        IF cOpSorTrd.sortrd-cde = Rowobject.sortrd-cde THEN
            v-enable = FALSE.

Cttrddo0.java
line: 4660
      query18.addComponent(new AdaptiveQuery(cop, "upper(cop.cteamCde) = ?", null, "cop.copId asc", new Object
[]
      {
         new character(ControlFlowOps.invokeDynamicFunctionIn("FcTeamCde", lhContainerSource))
      }, LockType.NONE));
```

#### #58 - 06/19/2014 04:46 AM - Marius Gligor

I created and converted a small test case using a WHERE clause and a DYNAMIC-FUNCTION operand.

```
FUNCTION f1 RETURNS CHARACTER ():
    RETURN "8".
END.

FIND FIRST customer WHERE customer > DYNAMIC-FUNCTION("f1").
   DISPLAY customer name.
PAUSE.
```

Here a snippet form AST tree which prove that the WHERE expression is equivalent to an IF expression.
Basically the generated path is the same EXPRESSION/GT. The first operand has many annotations which tell us that is a DB field.

```
<ast col="21" id="927712935962" line="5" text="WHERE" type="KW_WHERE">
   <annotation datatype="java.lang.Boolean" key="nested_ref" value="false"/>
   <ast col="0" hidden="true" id="927712935964" line="0" text="expression" type="EXPRESSION">
     <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
     <ast col="36" id="927712935965" line="5" text="&gt;" type="GT">
       <annotation datatype="java.lang.Long" key="matchtype" value="91"/>
       <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
       <ast col="27" id="927712935968" line="5" text="customer" type="FIELD_INT">
         <annotation datatype="java.lang.Long" key="oldtype" value="2381"/>
         <annotation datatype="java.lang.String" key="schemaname" value="p2j_test.customer.customer"/>
         <annotation datatype="java.lang.String" key="bufname" value="p2j_test.customer"/>
         <annotation datatype="java.lang.String" key="dbname" value="p2j_test"/>
         <annotation datatype="java.lang.Long" key="recordtype" value="12"/>
         <annotation datatype="java.lang.String" key="name" value="customer"/>
         <annotation datatype="java.lang.Long" key="type" value="373"/>
         <annotation datatype="java.lang.String" key="format" value="&quot;-&gt;,&gt;&gt;&gt;,&gt;&gt;9&quot;"
/>
         <annotation datatype="java.lang.String" key="fieldname" value="customer.customer"/>
         <annotation datatype="java.lang.Long" key="bufreftype" value="21"/>
         <annotation datatype="java.lang.String" key="uniquename" value="p2j_test.customer_p2j_test.customer"/
>
         <annotation datatype="java.lang.Long" key="refid" value="927712936000"/>
         <annotation datatype="java.lang.String" key="methodtxt" value="getCustomer"/>
         <annotation datatype="java.lang.Boolean" key="current_buffer" value="true"/>
         <annotation datatype="java.lang.Boolean" key="hql" value="true"/>
       </ast>
       <ast col="38" id="927712935971" line="5" text="DYNAMIC-FUNCTION" type="FUNC_POLY">
         <annotation datatype="java.lang.Long" key="oldtype" value="550"/>
         <annotation datatype="java.lang.Boolean" key="builtin" value="true"/>
         <annotation datatype="java.lang.Boolean" key="returnsunknown" value="true"/>
         <annotation datatype="java.lang.String" key="casttype" value="character"/>
         <annotation datatype="java.lang.String" key="poly_wrapper" value="integer"/>
         <annotation datatype="java.lang.Boolean" key="ignore_node" value="true"/>
         <annotation datatype="java.lang.Boolean" key="sub_expression" value="true"/>
         <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
         <annotation datatype="java.lang.String" key="param_modes" value=""/>
         <ast col="55" id="927712935972" line="5" text="&quot;f1&quot;" type="STRING">
           <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
           <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
         </ast>
       </ast>
     </ast>
   </ast>
</ast>
```

The Java converted code looks like:

```
    new FindQuery(customer, "customer.customer > ?", null, "customer.id asc", new Object[]
    {
       toUpperCase(new integer(ControlFlowOps.invokeDynamicFunction(character.class, "f1")))
    }).first();
```

The insertion of toUpperCase() function will generate a compile error.

The correct statement with casting should looks like:

```
new integer(ControlFlowOps.invokeDynamicFunction(character.class, "f1"))
```

or

```
new integer(toUpperCase(ControlFlowOps.invokeDynamicFunction(character.class, "f1")))
```

For this case the toUpperCase() is not necessary because the string is always converted to an integer value and the string case is not important in conversion.

#### #59 - 06/19/2014 08:24 AM - Marius Gligor

*- File mag_upd20140619a.zip added*

The rules for DYNAMIC-FUNCTION in WHERE expressions are like on any other expressions like IF for example.
So the type wrapping should be kept on all expressions in order to have the same behaviour like on P4GL.
To fix this issue I modified the convert/expressions.rules to emit a toUpperCase() method for FUNC_POLY cases if and only if the "poly_wrap" annotation is character or long character.
With this changes in place we could have:

new integer(ControlFlowOps.invokeDynamicFunction or
toUpperCase(new charcater(ControlFlowOps.invokeDynamicFunction or
toUpperCase(new longchar(ControlFlowOps.invokeDynamicFunction or
toUpperCase(ControlFlowOps.invokeDynamicFunction

toUpperCase() method is inserted when the following part of the AST tree is parsed. This is done after the expression is evaluated and the "poly_wrap" was created.
The FUNC_POLY node contains all annotations.

```
<ast col="0" id="927712936001" line="0" text="" type="QUERY_SUBST">
   <annotation datatype="java.lang.Long" key="peerid" value="932007903307"/>
   <ast col="0" id="927712936002" line="0" text="" type="EXPRESSION">
     <annotation datatype="java.lang.Long" key="peerid" value="932007903308"/>
     <ast col="38" id="927712936003" line="5" text="DYNAMIC-FUNCTION" type="FUNC_POLY">
       <annotation datatype="java.lang.Long" key="oldtype" value="550"/>
       <annotation datatype="java.lang.Boolean" key="builtin" value="true"/>
       <annotation datatype="java.lang.Boolean" key="returnsunknown" value="true"/>
       <annotation datatype="java.lang.String" key="casttype" value="character"/>
       <annotation datatype="java.lang.String" key="poly_wrapper" value="integer"/>
       <annotation datatype="java.lang.Boolean" key="ignore_node" value="true"/>
       <annotation datatype="java.lang.Boolean" key="sub_expression" value="true"/>
       <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
       <annotation datatype="java.lang.Long" key="oriref" value="927712935971"/>
       <annotation datatype="java.lang.String" key="param_modes" value=""/>
       <annotation datatype="java.lang.Boolean" key="ignorecast" value="true"/>
       <annotation datatype="java.lang.Long" key="peerid" value="932007903310"/>
       <ast col="55" id="927712936004" line="5" text="&amp;quot;f1&amp;quot;" type="STRING">
         <annotation datatype="java.lang.Boolean" key="is-literal" value="true"/>
         <annotation datatype="java.lang.Boolean" key="hql" value="false"/>
         <annotation datatype="java.lang.Long" key="peerid" value="932007903312"/>
       </ast>
     </ast>
```

```
   </ast>
</ast>
```

#### #60 - 06/20/2014 08:03 AM - Marius Gligor

*- File diff_2.txt added*

I've restarted an new customer server conversion using the latest changes. The full conversion took 5 hours and 21 minutes.
At the end the generated code is fully compiled (no errors) and the differences are the same (see the attached file diff_2.txt).
The differences are OK and expected because are the result of casting ("poly_wrapper") rule in expressions like EQUALS, NOT_EQ, GT, LT, GTE, LTE.
On all four cases the operands in expressions are of different type and the result of dynamic invocation DYNAMIC-FUNCTION is wrapped the the type of the other operand if this type is detected.
Note that on wrapping rule if the type of the operands are of the same type no wrapping is inserted in order to avoid unnecessary castings.

#### #61 - 06/20/2014 09:03 AM - Greg Shah

Eric will be reviewing and responding to this WHERE clause issue.

One thought: in the end, if your changes only change conversion for the customer's server project but not for MAJIC (AND the changes are conversion only), then your previous regression testing results will still be valid and you won't have to redo that.

#### #62 - 06/20/2014 09:47 AM - Eric Faulhaber

Marius, I've reviewed your previous several notes on the matter, and your reasoning and approach with respect to toUpperCase and WHERE clauses seems appropriate. Do you have any specific questions to which you still need answers?

#### #63 - 06/20/2014 09:56 AM - Marius Gligor

I have no other questions regarding conversion. I tried to do just punctual changes in rules and keep no changes on old rules as much as possible.
The customer's server code is a huge project having many complex P4GL statements which make him an ideal candidate for conversion tests.
Using the new designed rules for casting the result of DYNAMIC-FUNCTION return type produces expected results on conversion so I think that at this stage is OK.

#### #64 - 06/20/2014 03:59 PM - Greg Shah

Code Review 0619a

1. Line 1132 of expressions.rules has a typo.  uppercase = (cls  "caharacter" or cls  "longchar") should be uppercase = (cls  "character" or cls "longchar").

2. Can't we change this (in expressions.rules):

```
               <rule>polyfunc
                  <!-- FUNC_POLY case only if character or longchar -->
                  <rule>uppercase
                     <action>methodTxt = "toUpperCase"</action>
                     <action>charimp = true</action>
                     <action>
                        lastid = createJavaAst(java.static_method_call, methodTxt, lastid)
                     </action>
                  </rule>
                  <!-- other cases -->
                  <rule on="false">!isNote("casesens") or !getNoteBoolean("casesens")
                     <action>methodTxt = "toUpperCase"</action>
                     <action>charimp = true</action>
```

```
                    <action>
                       lastid = createJavaAst(java.static_method_call, methodTxt, lastid)
                    </action>
                 </rule>
              </rule>
```

into this:

```
              <!-- FUNC_POLY case only if character or longchar -->
              <rule>(polyfunc and uppercase) or !isNote("casesens") or !getNoteBoolean("casesens")
                 <rule>
                    <action>methodTxt = "toUpperCase"</action>
                    <action>charimp = true</action>
                    <action>
                       lastid = createJavaAst(java.static_method_call, methodTxt, lastid)
                    </action>
                 </rule>
              </rule>
```

**#65 - 06/23/2014 07:56 AM - Marius Gligor**

*- File test_cases.zip added*

*- File mag_upd20140623a.zip added*

1. Done.

2. Doesn't work should be:

```
<rule>(polyfunc and uppercase) or (!polyfunc and (!isNote("casesens") or !getNoteBoolean("casesens")))
```

3. However finally I've simplified the rules in expressions.rules using a single annotation "uppercase" which enable/disable insertion of toUpperCase() method.
By default the "uppercase" is true and the insertion of toUpperCase() is enabled in order to keep the original behaviour.
If the "poly_wrapper" annotation is present the insertion of toUpperCase() is enabled only if the cast is to type "character" or "longchar".
I tested the new rules on simple test cases (see attached files) and works good.
Also I started a new customer server conversion using the new rules which is currently running on my workstation.

#### #66 - 06/23/2014 11:37 AM - Greg Shah

Code Review 0623a

Your new version is much better.  The code looks good.

If the customer server conversion testing passes, please do run conversion regression testing for MAJIC.  You don't need to redo runtime regression testing since no runtime code was changed since your previous passing run.

#### #67 - 06/23/2014 12:42 PM - Marius Gligor

*- File diff_3.txt added*

The customer's server conversion passed. I have the same expected differences.
I'm going to prepare and start a MAJIC conversion regression test.

#### #68 - 06/23/2014 03:12 PM - Marius Gligor

MAJIC conversion regression test passed.

#### #69 - 06/23/2014 03:43 PM - Greg Shah

Please check in and distribute you update.

If I understand properly, both #2287 and #2050 can be closed now, right?

#### #70 - 06/23/2014 03:44 PM - Marius Gligor

Yes. All that remains is to commit the changes into remote repository and notify the team.

#### #71 - 06/25/2014 02:40 AM - Marius Gligor

*- % Done changed from 0 to 100*

*- Status changed from WIP to Review*

Passed MAJIC regression tests. Passed the customer server conversion test.
Committed revision 10551.

#### #72 - 06/25/2014 08:56 AM - Greg Shah

*- Status changed from Review to Closed*

#### #73 - 11/16/2016 12:06 PM - Greg Shah

*- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features*

### Files

| | | | |
|---|---|---|---|
| mag_upd20140602b.zip | 10.8 KB | 06/02/2014 | Marius Gligor |
| mag_upd20140603a.zip | 15 KB | 06/03/2014 | Marius Gligor |
| mag_upd20140603b.zip | 30.6 KB | 06/03/2014 | Marius Gligor |
| mag_upd20140606a.zip | 118 KB | 06/06/2014 | Marius Gligor |
| mag_upd20140610a.zip | 121 KB | 06/10/2014 | Marius Gligor |
| mag_upd20140611a.zip | 121 KB | 06/11/2014 | Marius Gligor |
| mag_upd20140612a.zip | 121 KB | 06/12/2014 | Marius Gligor |
| mag_upd20140613a.zip | 165 KB | 06/13/2014 | Marius Gligor |

| | | | |
|---|---|---|---|
| mag_upd20140613b.zip | 165 KB | 06/13/2014 | Marius Gligor |
| mag_upd20140616a.zip | 165 KB | 06/16/2014 | Marius Gligor |
| mag_upd20140617a.zip | 165 KB | 06/17/2014 | Marius Gligor |
| diff.txt | 1.38 KB | 06/18/2014 | Marius Gligor |
| mag_upd20140619a.zip | 177 KB | 06/19/2014 | Marius Gligor |
| diff_2.txt | 1.78 KB | 06/20/2014 | Marius Gligor |
| test_cases.zip | 2.29 KB | 06/23/2014 | Marius Gligor |
| mag_upd20140623a.zip | 177 KB | 06/23/2014 | Marius Gligor |
| diff_3.txt | 1.78 KB | 06/23/2014 | Marius Gligor |