Base Language - Feature #2089

INPUT function returning string value

03/12/2013 03:09 PM - Stanislav Lomany

Status:	Closed	**Start date:**	03/12/2013
Priority:	Normal	**Due date:**	
Assignee:	Stanislav Lomany	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	Conversion Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

History

#1 - 03/12/2013 03:12 PM - Stanislav Lomany

Copied from #2068:

CA
Another problem found in at least rtb/ folder:

```
def var i as int.

form i validate (length(i) > 0, "aa" ) with frame f1.

display i with frame f1.
```

converts to this validation expression:

```
   private class Validation0
   extends Validator
   {
      public logical validateExpression(BaseDataType _newValue_)
      {
         final integer _i = (integer) _newValue_;
         return isGreaterThan(length(_i), 0);
      }

      public character validateMessage(BaseDataType _newValue_)
      {
         return new character("aa");
      }
   }
```

I think when 4GL calls "length(i)" in the function it accesses the actual screen-value, as if the validation expression was:

```
form i validate (length(i:screen-value) > 0, "aa" ) with frame f1.
```

GES
In regard to note 57, this is essentially the same as:

```
form i validate (length(input i) > 0, "aa" ) with frame f1.
```

We currently do context analysis in ECW.expressionType() to try to handle the INPUT case, where it needs to morph its type into a character based on context.
Perhaps we can do this kind of context analysis for validation parameters too. We would need to add support to expressionType() and analyzeContext() for
reading the signatures of built-in functions, but we are doing that anyway...

CA
I don't think this is about the signature of the INPUT function. If we don't use the screen-value, then we don't compare what is actually in the widget, we will
use the value converted to the wrapper type, where an empty string means 0. Consider this:

```
def var i as int format ">>>>>>>>".

update i validate(length(i) = 0, "aaa").

message i.
```

If you don't enter anything, it will validate. If you put zero, it will not validate.

GES
What I mean is this:

1. In a validation expression, a reference to the current variable/field (that defined the associated widget) is an implicit use of the INPUT function.

2. When we process the INPUT function, we know the type of the variable/field that is being referenced, but if that type is not character, then we do context analysis on the containing node(s) to detect if we need to coerce the type to character. This is an undocumented behavior of the INPUT function
and we implement it, but our context analysis would fail in this case, unless we had the built-in function signature analysis implemented.

So, my point is that we should be treating these cases as an implicit INPUT function and doing the context analysis to ensure that all morphing/coercion
works like it does in the 4GL.

CA
I see your reasoning, but I don't think is correct. If the validation uses INPUT function:

```
update i validate(length(input i) = 0, "aaa").
```

then it converts to this, where there are still compile errors:

```
      public logical validateExpression(BaseDataType _newValue_)
      {
         return isEqual(length(frame0.getI()), 0); /* frame0.getI() returns an integer value, not character */
      }
   }
```

Even if we wrap the frame0.getI() in a character instance, this still does not give us what the user has actually typed.

If SCREEN-VALUE is used:

```
update i validate(length(i:screen-value) = 0, "aaa").
```

then the validation uses what the user has actually typed:

```
      public logical validateExpression(BaseDataType _newValue_)
      {
         final integer _i = (integer) _newValue_;
         return isEqual(length(frame0.widgeti().getScreenValue()), 0); /* getScreenValue() should return what
the user has actually typed */
      }
```

CA
Ok, I think our INPUT function is not working properly. Consider this:

```
def var i as int format ">>>>>>>>".

update i validate(length(i) >= 0, "aaa").

message i input i.
```

In 4GL it shows "0 " if nothing is entered. In P2J it shows "0 0".

LE: more, it looks like INPUT can return two types: the type of the widget and character, as this is possible:

```
def var ch as char.
def var i as int.
def var j as int.

form i with frame f1.

ch = input i. /* ok */
j = input i. /* ok */
ch = i. /* compile error */
```

#2 - 03/12/2013 03:46 PM - Stanislav Lomany

Greg, how should correctly converted validation code look like for the following example?

```
def var i as INT FORMAT ">>>>".

UPDATE i validate (length(i) > 0, "aa" ) with frame f1.
```

Should it look like this?

```
public logical validateExpression(BaseDataType _newValue_)
{
   return isGreaterThan(length(f1Frame.getScreenValue(f1Frame.widgeti())), 0);
}
```

BTW f1Frame.getScreenValue(f1Frame.widgeti()) returns "0" for the empty input field, but I guess that is incorrect.

#3 - 03/12/2013 04:05 PM - Greg Shah

As Constantin notes, the INPUT function in the 4GL can return either the actual data type of the widget OR it can return character. We already provide support for detecting when the return value of INPUT is used in places that require character. In such cases, we "override" the default type and return character instead. I am currently working to extend that facility to be more robust, including detecting the case of the length() builtin here.

When we override the type with character, we are implicitly using getScreenValue() which returns character. See builtin_functions.rules. Otherwise (when we are not overriding) we emit a specific getter for the widget, something like: frame.getI(). This will return integer in this case.

Inside a validation expression, we need to treat the reference to the widget lvalue as if it was an implicit INPUT function. So instead of processing this as "i" we need to process it as "INPUT i". As I note above, the backing facility for this is not fully available. To get an idea of how it works today, look at KW_INPUT in the ECW.expressionType(). The tricky part is that we have to figure out how we do this lookup implicitly, since there is no FUNC_POLY with oldtype == KW_INPUT as the parent node of the validation lvalue i (in this case).

We also need to know how badly broken our INPUT function (and widget:SCREEN-VALUE attribute) is at runtime.

#4 - 03/13/2013 01:47 PM - Stanislav Lomany

So, am I correct that except checking the runtime part we have two problems:
1. It is not properly recognized that an INPUT inside LENGTH should return a character value.
2. Validated lvalue should be represented as "INPUT lvalue" in a validation expression.

And I guess my task is only the second one?

#5 - 03/13/2013 01:53 PM - Constantin Asofiei

Stanislav Lomany wrote:

> So, am I correct that except checking the runtime part we have two problems:
> 1. It is not properly recognized that an INPUT inside LENGTH should return a character value.
> 2. Validated lvalue should be represented as "INPUT lvalue" in a validation expression.
>
> And I guess my task is only the second one?

The task is a combination of both. In a validation expression, you may have form i validate(length(i) <> 0, "bar) where def var i as int.

#6 - 03/13/2013 02:11 PM - Stanislav Lomany

Greg was saying that "I am currently working to extend that facility to be more robust, including detecting the case of the length() builtin here." so may be he is covering the first part?

#7 - 03/13/2013 02:12 PM - Constantin Asofiei

OK, I missed that. So your question remains.

#8 - 03/13/2013 04:42 PM - Stanislav Lomany

In regards to #2: should I parent all occurrences of a validated value with FUNC_POLY node as a solution?

#9 - 03/13/2013 06:00 PM - Greg Shah

Yes, you should work on item 2 and my changes will enable item 1. Please note that I don't want to transform all validation expressions into INPUT cases. The reason is that we already have the full parameter passing mechanism working for the common case. I think if we coulde detect the case where the problem will occur, that is the only place that we would want to rewrite as INPUT i.

#10 - 03/13/2013 06:14 PM - Greg Shah

About note 8: no.

You can do the following:

1. Determine the type (wrapper classname: like "integer") of the variable/field being validated. If the type is already character, then you are done.
2. Mark the node being tested with a boolean annotation named "implicit_input".
3. I will change ECW.expressionType() to treat this the same way as the INPUT function case. This means it will do context analysis to determine if the type needs to be coerced to character. If this is the case, then it will return character.
4. Call ECW.expressionType(node). If it returns character, then you know you have to coerce it.
5. Rewrite the node as the INPUT function (FUNC_POLY with oldtype KW_INPUT) as the parent. I think the conversion will naturally handle the rest for us.

You can write all of this now. It just won't work until I finish my changes. I will try to get them in tonight but it may be tomorrow morning if there are any regressions.

#11 - 03/13/2013 09:55 PM - Greg Shah

I have an update posted in #2053. It is in testing, but hopefully you can use it to test your changes.

#12 - 03/14/2013 05:07 PM - Greg Shah

```
    If a node has the "implicit_input" annotation it will case infinite cycle:
```

```
   public static String expressionType(Aast source, boolean infer)
   throws IllegalArgumentException,
          UnsupportedOperationException
   {
     ....
```

```
      // special case for implicit INPUT function support
      if (source.isAnnotation("implicit_input"))
      {
         return processInputBuiltinFunc(source, infer);
      }
```

```
...
```

```
   private static String processInputBuiltinFunc(Aast source, boolean infer)
   {
      // normal return type is the same as the referent
      String jcls = expressionType(source, false);

      ...
```

On the 1st entry into processInputBuiltinFunc(), set a static flag (named something like inInputFunc) that will bypass the 2nd call (and subsequent calls) to processInputBuiltinFunc() (if (source.isAnnotation("implicit_input") && !inInputFunc)).

#13 - 03/18/2013 11:25 AM - Stanislav Lomany

Signature of the LENGTH function stored into SignatureHelper.map_func map is { ParmType.BDT, ParmType.CHAR }. As the first parameter LENGTH can take char or raw, so I decided to change the signature to multiple signatures: the first taking char and the second taking raw, but in this case ECW.resolveSignature returns no signature (because none of the signatures match passed integer). May be we should have a "default" signature which is returned in the case if none of the multiple signatures match? (That is actually how it works for a single-signature case.)

#14 - 03/18/2013 12:19 PM - Greg Shah

What is the problem with the use of ParmType.BDT (BaseDataType) as the 1st parameter (the way it was originally coded)?

#15 - 03/18/2013 01:16 PM - Stanislav Lomany

> What is the problem with the use of ParmType.BDT (BaseDataType) as the 1st parameter (the way it was originally coded)?

In this case when we pass integer to LENGTH it is not detected that coercion to character is required.

#16 - 03/18/2013 01:34 PM - Greg Shah

Please post the code you are using to implement the implicit_input analysis.

#17 - 03/18/2013 01:50 PM - Greg Shah

In regard to your idea about a default signature, I am OK with that approach. How about this: if the signature matching algorithm find more than 1 match (based on the actual signature being used and mandatory # of parameters provided), then it should return the first match in the list (which will be the default). Thus, you can create 2 signatures for LENGTH and make the character one the default by placing it first.

#18 - 03/18/2013 02:14 PM - Stanislav Lomany

> Please post the code you are using to implement the implicit_input analysis.

```
...
<action>putNote("implicit_input", true)</action>
<action>val = ecw.expressionType(copy)</action>

<rule>val.equals("character")
   <action>
      ref = createProgressAst(prog.func_poly,
                              copy.parent,
                              copy.indexPos)
   </action>
   ...
```

#19 - 03/18/2013 02:59 PM - Greg Shah

OK, you have to some some additional things here.

1. Before you set the implicit_input, you must first know the default type that would be returned. In this length(i) case, the type would be "integer". So you must call expressionType() without the "implicit_input" annotation. Let's call this the "default type".

2. The coercion should only occur if the default type is not equal to the implicit_input type (AND the implicit_input type is "character"). In other words, if the default type being returned is already "character", then no coercion is needed.

3. I assume that you are setting oldtype == "KW_INPUT" and re-parenting copy at the new node.

#20 - 03/18/2013 03:07 PM - Stanislav Lomany

Greg Shah wrote:

> OK, you have to some some additional things here.
>
> 1. Before you set the implicit_input, you must first know the default type that would be returned. In this length(i) case, the type would be "integer". So you must call expressionType() without the "implicit_input" annotation. Let's call this the "default type".
>
> 2. The coercion should only occur if the default type is not equal to the implicit_input type (AND the implicit_input type is "character"). In other words, if the default type being returned is already "character", then no coercion is needed.
>
> 3. I assume that you are setting oldtype == "KW_INPUT" and re-parenting copy at the new node.

Please check the full code, I think it matches these requirements:

```
<function name="handle_implicit_input">
   <variable   name="val"       type="java.lang.String" />
   <variable   name="vnode"     type="com.goldencode.ast.Aast" />
   <variable   name="vtarget"   type="com.goldencode.ast.Aast" />

   <rule>vnode = getAncestorOfType(prog.kw_validate)</rule>
   <rule>vtarget = vnode.getFirstChild()</rule>

   <rule>val = #(java.lang.String) vnode.getAnnotation("wrapper")</rule>

   <rule>!val.equals("character")

      <action>putNote("implicit_input", true)</action>
      <action>val = ecw.expressionType(copy)</action>

      <rule>val.equals("character")
         <action>
            ref = createProgressAst(prog.func_poly,
                                    copy.parent,
                                    copy.indexPos)
         </action>
         <action>ref.putAnnotation("oldtype", #(long) prog.kw_input)</action>
         <action>ref.putAnnotation("frame-id",
                        #(long) vtarget.getAnnotation("frame-id"))</action>
         <action>ref.putAnnotation("builtin", true)</action>
         <action>ref.putAnnotation("returnsunknown", false)</action>
         <action>ref.putAnnotation("override", true)</action>

         <action>copy.remove()</action>
         <action>ref.graft(copy)</action>
         <action>putNote("getter",
                         #(java.lang.String) vtarget.getAnnotation("getter"))</action>
         <action>putNote("setter",
                         #(java.lang.String) vtarget.getAnnotation("setter"))</action>
         <action>putNote("javaname",
                         #(java.lang.String) vtarget.getAnnotation("javaname"))</action>
         <action>putNote("accessor",
                         #(java.lang.String) vtarget.getAnnotation("accessor"))</action>
         <action>putNote("widgettype",
                         #(java.lang.String) vtarget.getAnnotation("widgettype"))</action>
         <action>putNote("frame-id",
                         #(long) vtarget.getAnnotation("frame-id"))</action>
      </rule>
   </rule>
</function>
```

#21 - 03/18/2013 03:13 PM - Stanislav Lomany

> In regard to your idea about a default signature, I am OK with that approach. How about this: if the signature matching algorithm find more than 1 match (based on the actual signature being used and mandatory # of parameters provided), then it should return the first match in the list (which will be the default). Thus, you can create 2 signatures for LENGTH and make the character one the default by placing it first.

OK, in our case: we have two definitions (CHAR/RAW) and a single passed INTEGER. Search based on function parameter types (using getSignature(int kw, int type, String[] ptypes)) fails (i.e. no matches). What should we do at this point? My suggestion is to pick the first function which matches (int kw, int type). Not sure how the number of mandatory parameters will help us.

#22 - 03/18/2013 03:36 PM - Greg Shah

Yes, that is my point. getSignature(int, int, String[]) should be modified to return the default as the first one defined IF it cannot otherwise find a match AND there are multiple signatures. Please consider that the number of actual parameters vs the number of parameters in the signature will also affect the default. This requires that mand be used.

The idea is that if you pass 4 parameters to the DATETIME function, but the type of the parameters doesn't exactly match, then we know that we will have to return a default signature. BUT we must only return a signature that could possibly match, which means that the DATETIME and DATETIME cannot match, but since there is a signature DATETIME with only 4 mandatory parms, that would be the default. If there was a 2nd possible "fuzzy" match, then the first one defined would be returned.

Also: this fuzzy matching may have bad side effects on other places we use expressionType(). For this reason, I think we might want to add a flag to the main expressionType(AST, boolean) signature to make it expressionType(AST, boolean, boolean). The 3rd parm would be to enable "fuzzy" signature matching, but it would only be set to true in your use case.

#23 - 03/18/2013 05:27 PM - Stanislav Lomany

> Also: this fuzzy matching may have bad side effects on other places we use expressionType(). For this reason, I think we might want to add a flag to the main expressionType(AST, boolean) signature to make it expressionType(AST, boolean, boolean). The 3rd parm would be to enable "fuzzy" signature matching, but it would only be set to true in your use case.

Please note that this flag will also be added to analyzeContext and parameterTypeName functions.

#24 - 03/18/2013 05:35 PM - Greg Shah

Understood.

#25 - 03/19/2013 07:51 AM - Constantin Asofiei

Greg/Stanislav: we can make this a low priority, as after the latest schema VALEXP changes in #1620, the problematic VALEXPS are no longer showing errors (unless the errors are hidden by other compile errors). The idea is I think the VALEXPs were generated too aggresively (it included the field references in FORM statement, which was not needed).

#26 - 03/19/2013 09:17 AM - Greg Shah

Actually, this should be pretty much done at this point. Let's finish it now.

Stanislav: when will the update be ready?

#27 - 03/19/2013 09:42 AM - Stanislav Lomany

> Stanislav: when will the update be ready?

I'm writing javadocs / merging it.

#28 - 03/19/2013 02:58 PM - Stanislav Lomany

- File svl_upd20130319a.zip added

Not yet regression-tested.

#29 - 03/19/2013 03:13 PM - Greg Shah

The code looks good. I will put it into conversion testing here since it will be faster.

#30 - 03/19/2013 07:46 PM - Greg Shah

1 compile problem in Majic:

```
compile:
    [javac] Compiling 14607 source files to /home/ges/timco/build/classes
    [javac] /home/ges/timco/src/aero/timco/majic/train2/Rst01.java:1922: error: method getScreenValue in inter
face CommonFrame cannot be applied to given types;
    [javac]                 return isEqual(fataFrame.getScreenValue(), fataFrame.widgetCurCustNum().getScreenVa
lue(), "?");
    [javac]                                         ^
    [javac]   required: GenericWidget
    [javac]   found: no arguments
    [javac]   reason: actual and formal argument lists differ in length
```

And there were these diffs:

```
diff -r generated/20130319a/src/aero/timco/majic/codes/Taxcode.java generated/20130319b/src/aero/timco/majic/c
odes/Taxcode.java
1280,1281c1280
```

```
<          final logical _taxFreight = (logical) _newValue_;
<          return or(isEqual(_taxFreight, "yes"), new LogicalExpression()
---
>          return or(isEqual(fTaxcodeFrame.getScreenValue(fTaxcodeFrame.widgetTaxFreight()), "yes"), new Logic
alExpression()
1285c1284
<                return isEqual(_taxFreight, "no");
---
>                return isEqual(fTaxcodeFrame.getScreenValue(fTaxcodeFrame.widgetTaxFreight()), "no");
diff -r generated/20130319a/src/aero/timco/majic/job/Empcal.java generated/20130319b/src/aero/timco/majic/job/
Empcal.java
2154c2154
<          return or(isNotEqual(_tDate, ""), isUnknown(_tDate));
---
>          return or(isNotEqual(fScreen5Frame.getScreenValue(fScreen5Frame.widgetTDate()), ""), isUnknown(_tDa
te));
diff -r generated/20130319a/src/aero/timco/majic/job/OtanalysisR.java generated/20130319b/src/aero/timco/majic
/job/OtanalysisR.java
1580c1580
<          return concat("INVALID SO #.  PLEASE ENTER SO # >= ", valueOf(fInputsFrame.getScreenValue(fInputsFr
ame.widgetSCoNum())));
---
>          return concat("INVALID SO #.  PLEASE ENTER SO # >= ", valueOf(fInputsFrame.getSCoNum()));
1608c1608
<          return concat("INVALID EMPLOYEE #.  PLEASE ENTER EMPLOYEE # >= ", valueOf(fInputsFrame.getScreenVal
ue(fInputsFrame.widgetSEmpNum())));
---
>          return concat("INVALID EMPLOYEE #.  PLEASE ENTER EMPLOYEE # >= ", valueOf(fInputsFrame.getSEmpNum()
));
1636c1636
<          return concat("INVALID DATE.  PLEASE ENTER DATE >= ", valueOf(fInputsFrame.getScreenValue(fInputsFr
ame.widgetSDate())));
---
>          return concat("INVALID DATE.  PLEASE ENTER DATE >= ", valueOf(fInputsFrame.getSDate()));
diff -r generated/20130319a/src/aero/timco/majic/train2/EmpTransfer.java generated/20130319b/src/aero/timco/ma
jic/train2/EmpTransfer.java
1034,1035c1034
<          final integer _iOldEmployee = (integer) _newValue_;
<          return isNotEqual(_iOldEmployee, "");
---
>          return isNotEqual(fInFrame.getScreenValue(fInFrame.widgetIOldEmployee()), "");
1049,1050c1048
<          final integer _iNewEmployee = (integer) _newValue_;
<          return isNotEqual(_iNewEmployee, "");
---
>          return isNotEqual(fInFrame.getScreenValue(fInFrame.widgetINewEmployee()), "");
diff -r generated/20130319a/src/aero/timco/majic/train2/Rst01.java generated/20130319b/src/aero/timco/majic/tr
ain2/Rst01.java
1922c1922
<                return isEqual(fataFrame.widgetCurCustNum().getScreenValue(), "?");
---
>                return isEqual(fataFrame.getScreenValue(), fataFrame.widgetCurCustNum().getScreenValue(), "?"
);
```

Please check these diffs carefully. The 20130319b/ directory is the one that this update created. The 20130319a/ directory is a baseline from the currently checked in bzr sources.

#31 - 03/20/2013 01:49 PM - Stanislav Lomany

Note that

```
UPDATE i VALIDATE (i > 0, INPUT i) WITH FRAME f1.
```

is incorrectly converted to

```
public character validateMessage(BaseDataType _newValue_)
{
   return f1Frame.getI();    // should be string, not integer
}
```

#32 - 03/20/2013 02:00 PM - Greg Shah

When we search upward for KW_VALIDATE and enable implicit input testing, we must ensure that we are a descendant of the 1st child of the KW_VALIDATE...

#33 - 03/20/2013 02:13 PM - Stanislav Lomany

> When we search upward for KW_VALIDATE and enable implicit input testing, we must ensure that we are a descendant of the 1st child of the KW_VALIDATE

Actually implicit input processing is correctly applied to both VALIDATE parameters.

Anyway, should I fix the explicit INPUT edge case posted above?

#34 - 03/20/2013 02:16 PM - Greg Shah

Yes, please do.

#35 - 03/21/2013 06:55 AM - Stanislav Lomany

- File svl_upd20130319a.zip added

#36 - 03/21/2013 06:57 AM - Stanislav Lomany

- File implicit-input.p added

#37 - 03/21/2013 08:12 AM - Greg Shah

Code Review:

It looks good. 1 minor issue: the comment was corrupted in "handle_implicit_input" in the latest common-progress when you merged.

Also: please name the update for the current day (svl_upd20130321a.zip) instead of reusing the same filename from before.

Please fix these things, re-upload and I will conversion test it.

#38 - 03/21/2013 08:42 AM - Stanislav Lomany

- File svl_upd20130321a.zip added

#39 - 03/21/2013 09:51 AM - Greg Shah

Here are the results of the conversion testing:

```
diff -r generated/20130319c/src/aero/timco/majic/codes/Taxcode.java generated/20130321a/src/aero/timco/majic/c
odes/Taxcode.java
1280,1281c1280
<          final logical _taxFreight = (logical) _newValue_;
<          return or(isEqual(_taxFreight, "yes"), new LogicalExpression()
---
>          return or(isEqual(fTaxcodeFrame.getScreenValue(fTaxcodeFrame.widgetTaxFreight()), "yes"), new Logic
alExpression()
1285c1284
<                return isEqual(_taxFreight, "no");
---
>                return isEqual(fTaxcodeFrame.getScreenValue(fTaxcodeFrame.widgetTaxFreight()), "no");
diff -r generated/20130319c/src/aero/timco/majic/job/Empcal.java generated/20130321a/src/aero/timco/majic/job/
Empcal.java
2154c2154
<          return or(isNotEqual(_tDate, ""), isUnknown(_tDate));
---
>          return or(isNotEqual(fScreen5Frame.getScreenValue(fScreen5Frame.widgetTDate()), ""), isUnknown(_tDa
te));
diff -r generated/20130319c/src/aero/timco/majic/job/OtanalysisR.java generated/20130321a/src/aero/timco/majic
/job/OtanalysisR.java
1580c1580
<          return concat("INVALID SO #.  PLEASE ENTER SO # >= ", valueOf(fInputsFrame.getScreenValue(fInputsFr
ame.widgetSCoNum())));
---
>          return concat("INVALID SO #.  PLEASE ENTER SO # >= ", valueOf(fInputsFrame.getSCoNum()));
1608c1608
<          return concat("INVALID EMPLOYEE #.  PLEASE ENTER EMPLOYEE # >= ", valueOf(fInputsFrame.getScreenVal
ue(fInputsFrame.widgetSEmpNum())));
---
>          return concat("INVALID EMPLOYEE #.  PLEASE ENTER EMPLOYEE # >= ", valueOf(fInputsFrame.getSEmpNum()
));
1636c1636
<          return concat("INVALID DATE.  PLEASE ENTER DATE >= ", valueOf(fInputsFrame.getScreenValue(fInputsFr
ame.widgetSDate())));
---
>          return concat("INVALID DATE.  PLEASE ENTER DATE >= ", valueOf(fInputsFrame.getSDate()));
diff -r generated/20130319c/src/aero/timco/majic/train2/EmpTransfer.java generated/20130321a/src/aero/timco/ma
jic/train2/EmpTransfer.java
1034,1035c1034
<          final integer _iOldEmployee = (integer) _newValue_;
<          return isNotEqual(_iOldEmployee, "");
---
>          return isNotEqual(fInFrame.getScreenValue(fInFrame.widgetIOldEmployee()), "");
1049,1050c1048
<          final integer _iNewEmployee = (integer) _newValue_;
<          return isNotEqual(_iNewEmployee, "");
---
>          return isNotEqual(fInFrame.getScreenValue(fInFrame.widgetINewEmployee()), "");
```

These changes look OK to me. Please confirm here whether or not they are all expected. If they are all expected you can check in and distribute the change.

#40 - 03/21/2013 10:14 AM - Stanislav Lomany

Yes, they all are OK.

#41 - 03/21/2013 10:37 AM - Greg Shah

Please do check in and distribute the change.

#42 - 03/21/2013 11:23 AM - Greg Shah

- Status changed from WIP to Closed

- Target version set to Milestone 4

#43 - 11/16/2016 11:06 AM - Greg Shah

- Target version changed from Milestone 4 to Conversion Support for Server Features

Files

svl_upd20130319a.zip	74.9 KB	03/19/2013	Stanislav Lomany
svl_upd20130319a.zip	75.3 KB	03/21/2013	Stanislav Lomany
implicit-input.p	999 Bytes	03/21/2013	Stanislav Lomany
svl_upd20130321a.zip	75.3 KB	03/21/2013	Stanislav Lomany