

User Interface - Feature #1791

implement dynamic UI support

10/30/2012 09:43 AM - Greg Shah

<b>Status:</b>	Closed	<b>Start date:</b>	02/21/2013
<b>Priority:</b>	Normal	<b>Due date:</b>	11/08/2013
<b>Assignee:</b>		<b>% Done:</b>	100%
<b>Category:</b>		<b>Estimated time:</b>	0.00 hour
<b>Target version:</b>	GUI Support for a Complex ADM2 App	<b>vendor_id:</b>	GCD
<b>billable:</b>	No		
<b>Description</b>			
<b>Subtasks:</b>			
Feature # 2025: conversion support for the CREATE_WIDGET statement			Closed
Feature # 2026: conversion support for CREATE_BROWSE			Closed
Feature # 2027: preliminary runtime support for CREATE_WIDGET and an IN WINDOW clause			Closed
Feature # 2028: complete runtime support for dynamic UI			Closed
<b>Related issues:</b>			
Related to User Interface - Feature #2018: runtime support for CURRENT-WINDOW...		Closed	08/06/2013 08/13/2013
Related to User Interface - Feature #1789: implement the toggle-box widget		Closed	02/10/2015
Related to User Interface - Bug #2396: memory leak when using shared frames		New	
Related to User Interface - Feature #3246: reduce the amount of data being se...		Closed	

History

#1 - 10/30/2012 09:44 AM - Greg Shah

CREATE WIDGET (for static widget types and for the case of VALUE), CREATE BROWSE, DELETE WIDGET

#2 - 10/31/2012 02:53 PM - Greg Shah

- Target version set to Milestone 12

#3 - 02/21/2013 08:00 PM - Greg Shah

High priority for milestone 4:

```
CREATE WINDOW my-handle-var
  ASSIGN attr1 = expr1
    attr2 = expr2.
```

I think this can be handled by a factory method (which returns a WrappedResource that is assigned to the named handle var) and the widget type can be passed as the parameter. Then the attribute assignments should emit after that, implicitly dereferencing the same handle.

The CREATE BROWSE can be left alone for now.

#### #4 - 02/22/2013 05:06 AM - Constantin Asofiei

The conversion rules to emit this were pretty simple (I have code for all except CREATE BROWSE), although I'm not sure how to name the factory class, as WidgetFactory is already taken by a p2j.ui.client.driver class, and if I add another one in p2j.ui.WidgetFactory, I will need to disambiguate in lots of classes, so I chose CreateWidgetFactory for now. The same for runtime part, was pretty straightforward too, just instantiate a p2j.ui.\*Widget, depending on the text.

Now, the hard part. The ASSIGN clause can set the widget's attributes before it was attached to a frame. Thus, the P2J runtime will fail to set the attribute using the handle, as when the unwrap is done, the contained widget is not valid, as it was not attached to a frame. More, 4GL allows setting the attributes even outside the ASSIGN clause, before the widget was attached to a frame, as in:

```
create button h.  
h:row = 1.  
h:column = 1.
```

For this, I think we need to change GenericWidget.isValid to return true if is not attached to a frame. ~~More, we will need this runtime change to be released together with the conversion change, as MAJIC uses CREATE WINDOW statement.~~ I see that all this code is commented out, but nevertheless, I will leave the protection code on.

#### #5 - 02/22/2013 09:15 AM - Greg Shah

Yes, I like the approach. The assignments for a CREATE WIDGET can be easily dealt with by implicitly using the handle (or if the code explicitly uses the handle too). So with your protection code on isValid() I think we are OK. FYI, the customer code uses this heavily in GUI and a little bit in the server code, so that is why we definitely need to support it.

As for the name, how about DynamicWidgetFactory?

#### #6 - 02/22/2013 10:11 AM - Constantin Asofiei

- File ca\_upd20130222f.zip added

Attached update adds CREATE WIDGET support (explicit and VALUE versions). Converted code looks like:

```
def var h as handle.  
def var ch as char.  
create window h assign column = 1 row = 1.  
create value(ch).  
create value("WINDOW").
```

java code:

```
DynamicWidgetFactory.createWindow(h);  
handle.unwrapWidget(h).setColumn(1);
```

```
handle.unwrapWidget(h).setRow(1);

DynamicWidgetFactory.create((ch).toStringMessage(), h);
DynamicWidgetFactory.create("WINDOW", h);
```

If IN WIDGET-POOL clause is present, there are API versions which accept this too.

#### #7 - 02/22/2013 10:35 AM - Constantin Asofiei

The trigger phrase was not touched - should I work on this too ?

#### #8 - 02/22/2013 12:52 PM - Greg Shah

No, leave the trigger phrase out for now.

2 questions:

1. Why not do everything using an enum: WidgetTypes.WINDOW, WidgetTypes.FILLIN...

```
CREATE WINDOW h.
```

Would be:

```
DynamicWidgetFactory.create(WidgetTypes.WINDOW, h);
```

2. Why not statically convert String literal widget names into the well known match:

```
CREATE VALUE("WINDOW") h.
```

This is the same as this, right?

```
CREATE WINDOW h.
```

**#9 - 02/22/2013 01:07 PM - Constantin Asofiei**

1. Why not do everything using an enum: WidgetTypes.WINDOW, WidgetTypes.FILLIN...

In the end, we will still have to match a text, as the worst case scenario is when a character var/complex expression is used with CREATE VALUE(...). I chose the approach of having dedicated APIs because IMO is cleaner, but I can switch to only one API DynamicWidgetFactory.create API which accepts an enum constant.

2. Why not statically convert String literal widget names into the well known match:  
This is the same as this, right?

Yes, they are the same. I will disambiguate the string literal, if it is a known widget type. Is it OK to do it in the pre-annotation phase (in fixups or progress.g) - I'm thinking of directly creating the widget-type AST, as if CREATE WINDOW was used instead of CREATE VALUE.

**#10 - 02/22/2013 01:15 PM - Greg Shah**

I chose the approach of having dedicated APIs because IMO is cleaner, but I can switch to only one API DynamicWidgetFactory.create API which accepts an enum constant.

You are probably right that the result is better this way. It is just a big more work on the conversion side. Go with your approach.

Is it OK to do it in the pre-annotation phase (in fixups or progress.g) - I'm thinking of directly creating the widget-type AST, as if CREATE WINDOW was used instead of CREATE VALUE.

No, please do this after post\_parse\_fixups.

**#11 - 02/22/2013 01:42 PM - Constantin Asofiei**

I chose the approach of having dedicated APIs because IMO is cleaner, but I can switch to only one API `DynamicWidgetFactory.create` API which accepts an enum constant.

You are probably right that the result is better this way. It is just a bit more work on the conversion side. Go with your approach.

OK, so I'll drop the enum idea.

Is it OK to do it in the pre-annotation phase (in fixups or progress.g) - I'm thinking of directly creating the widget-type AST, as if `CREATE WINDOW` was used instead of `CREATE VALUE`.

No, please do this after `post_parse_fixups`.

OK, I'm working on it. Let me know if you want me to merge it with [#2013](#) SELF update (I mean, if you want to put it into testing later tonight).

**#12 - 02/22/2013 01:47 PM - Greg Shah**

Yes, please do merge it with the [#2013](#) update. We will put them both in together.

**#13 - 02/22/2013 02:18 PM - Constantin Asofiei**

- File `ca_upd20130222i.zip` added

Merged with [#2013](#), fixed conversion of `CREATE VALUE(type)` when the type is a known widget type.

**#14 - 02/22/2013 04:37 PM - Greg Shah**

- File `ca_upd20130222k.zip` added

Merged up to bzip 10188. This version will be going into testing next.

**#15 - 02/22/2013 10:13 PM - Greg Shah**

Passed conversion testing. Checked into bsr as 10190.

**#16 - 04/18/2013 12:53 PM - Constantin Asofiei**

What should [#2027](#) "preliminary runtime support" contain? Do you mean to avoid implementing the exotic cases in this task?

For the runtime implementation of dynamic widgets, I see these major points:

1. the dynamic widget creation is already added (by DynamicWidgetFactory) - so nothing to do to actually create the widget
2. the pool behavior is not implemented - should be done for [#2027](#); estimate is 20 hours for this
3. the widget deletion I don't think is properly implemented - this is linked to pooling, should be done for [#2027](#); another 20 hours estimate
4. determine if is possible to move widgets from a frame to another frame (if a handle to the widget is obtained) - this can be done in [#2028](#)
5. for these, the estimate for [#2027](#) is 40 hours; this does not include implementing new widgets

For the IN WINDOW clause, things get more complicated, as we need to determine:

1. I think the WINDOW attribute is highly related to this
2. if this clause is missing, does it assume that CURRENT-WINDOW is used? or it uses the WINDOW attribute?
3. what happens just after a frame is defined, using DEFINE FRAME, CREATE FRAME or FORM statement (i.e. a statement which doesn't actually display the frame): does this attribute get set to something?
4. the estimate for this at this time is somewhat a shot in the dark, so my best guess is 40 hours for preliminary support (please explain what [#2027](#) should not include)

**#17 - 04/18/2013 01:17 PM - Greg Shah**

Do you mean to avoid implementing the exotic cases in this task?

Yes. I mean for [#2027](#) to be the minimum needed for the server code to work properly. I don't have a problem with the work listed above (including the 4GL research). It seems reasonable.

Then we will complete the runtime work in [#2028](#) for the GUI project.

I will go with 60 hours as the estimate.

**#18 - 11/11/2013 03:15 PM - Vadim Gindin**

Constantin Asofiei wrote:

What should [#2027](#) "preliminary runtime support" contain? Do you mean to avoid implementing the exotic cases in this task?

..

1. the pool behavior is not implemented - should be done for [#2027](#); estimate is 20 hours for

Could you please describe more concrete what do you mean here? As I can see DynamicWidgetFactory already contains pool logic in create(..) methods.

**#19 - 11/11/2013 03:42 PM - Constantin Asofiei**

Vadim Gindin wrote:

1. the pool behavior is not implemented - should be done for [#2027](#); estimate is 20 hours for

Could you please describe more concrete what do you mean here? As I can see DynamicWidgetFactory already contains pool logic in create(..) methods.

The widget pooling is implemented (i.e. widgets get added/deleted properly, when pools are used); what you need to focus on is what happens when a dynamic widget gets deleted (how is the UI affected, same for frame layout, etc).

**#20 - 11/19/2013 04:42 PM - Vadim Gindin**

- *File err.log added*

Well I tried to concentrate on what will happen with UI if some dynamic widget is deleted and wrote some test procedure:

```
def var wh as widget-handle.  
def button b_quit label "Quit" AUTO-ENDKEY.  
def button b_make label "Make button".  
def button b_del label "Delete button".  
  
def frame butt-frame
```

```

    b_make b_del b_quit
    with centered size 40 by 5 side-labels.

on choose of b_make in frame butt-frame
do:
    create button wh.

    assign wh:frame = frame butt-frame:handle
        wh:column = 4
        wh:label = "Dynamic button"
        wh:sensitive = true
        wh:visible = true
        wh:row = 2.
end.

on choose of b_del in frame butt-frame
do:
    delete widget wh.
end.
enable all with frame butt-frame.

wait-for choose of b_quit in frame butt-frame.

```

It shows 3 buttons: to create dynamic button, delete it and quit. I converted this procedure and converted Java class failed during execution of the make button trigger i.e. dynamic button creation failed (see err.log). I also debugged the problem and found out that `ThinClient.enable(..)` is called with `widgetId=-1`. I suspect that is the problem: `widgetId` is not transferred.. I don't know could you help me with it?

I also tried to analyze existing procedures linked with dynamic widgets creation. I found some amount of procedures related to `widget_pool`. In these procedures there are only simple form of create button is used. Simple - without any attributes. It seems that the real case of dynamic button creation is not really implemented or good tested. Do I wrong?

I also found, that the statement "create button .. assign" is not supported by rules (corresponding converted Java class had compile error). After that I was forced to separate button creation and attributes filling.

And at the final of this comment I need the common advice. Bearing in mind your experience with UI testing I want to ask: how to test effectively UI changes after some action? Should I track positions of elements and how it changed or there is some other way, more nice?

## #21 - 11/20/2013 11:12 AM - Constantin Asofiei

Vadim Gindin wrote:

It seems that the real case of dynamic button creation is not really implemented or good tested. Do I wrong?

You are correct; this is what [#2027](#) task is about: a first pass at the runtime for dynamic widgets.

Before starting working on how dynamic widgets should be implemented, you need to understand how the static frames/widgets work. Look into `GenericFrame.createFrame`, how the interfaces for a static frame look and what `GenericFrame.pushScreenDefinitions` and `WidgetRegistry.pushDefinition` do, when a new frame is being registered. You will notice that a widget is implemented on both sides: on the server-side a widget will have a `GenericWidget` instance associated with it and on the client-side there will be a corresponding `p2j.ui.client.widget.Widget` implementation (see `AbstractWidget` too).

You will need to determine a way to interact with the client-side and push or remove dynamic widgets and dynamic frames.

## #22 - 11/20/2013 12:32 PM - Vadim Gindin

I thought that I can write more simple test to check the support of DELETE statement, i.e. I don't need to create button with assigning properties to created button (with showing it on the screen). Here this procedure:

```
def var delwh as widget-handle.

/* default pool - session pool */
if delwh <> ? then
    message "Strange error: delwh have created and not initialized but not empty: " delwh.

create button delwh.
if delwh = ? then
    message "Error: delwh is ?".

delete widget delwh.
if valid-handle(delwh) then
    message "Error: delwh is not empty: " delwh.

/* specific created pool */
create widget-pool "somePool".

create button delwh in widget-pool "somePool".
if delwh = ? then
    message "Error: delwh is ?".

delete widget delwh.
if valid-handle(delwh) then
    message "Error: delwh is not empty: " delwh.

/* delete widget for invalid handle */
create button delwh in widget-pool "somePool".
if delwh = ? then
    message "Error: delwh is ?".

delete widget-pool "somePool".

delete widget delwh.
if valid-handle(delwh) then
    message "Error: delwh is not empty: " delwh.
```

I tried to write it in a Constantin's manner: output messages only in cases of error. It is so except the last case. There is an error in Progress: firstly create a button in a pool, deleted pool and trying to delete this button after that. When the pool is deleted the Progress automatically deletes all widgets in it, so following DELETE WIDGET (button) statement raises the error Invalid handle. Not initialized or points to a deleted object (3135).

In other cases (first two - create/delete button in default or specified pool) the Progress don't output anything, that means - button successfully deleted.

Converted class works similar: in first to cases it outputs nothing and in third case it outputs the error. The only error message differs: Invalid or inappropriate handle value given to DELETE OBJECT or DELETE PROCEDURE statement (5425). Should I correct the error message in this case?

This procedure shows that DELETE WIDGET statement is implemented correctly (except the error message). Should I try some additional tests to check it?

**#23 - 11/20/2013 02:02 PM - Constantin Asofiei**

Should I correct the error message in this case?

No, do not focus on it; this is a conversion issue (to show the proper message, we need to add some boolean flag that DELETE WIDGET was used, as we distinguish between DELETE WIDGET and DELETE OBJECT only by the number of passed widgets). For the DELETE WIDGET wid1 wid2 wid3 case, the error messages are OK.

The widget-pool task [#1792](#) did not test what happens **during** the resource deletion; all tests were done to make sure the pooled resources get deleted at the appropriate time, when widget-pools are used.

I appreciate encoding the expected behaviour within the tests, but I think you will not find this approach useful for what it expects you; to test how the UI behaves when widgets are added dynamically, you will have to record somewhere how the UI looks after the dynamic widget was added, after was deleted, etc, and this for every kind of widget. Also, you will have to think how to test the dynamic frame layout (i.e. a widget is added in the middle of the frame - how is the layout affected?). From all these tests, you will need to determine some rules describing how the frame is affected.

Greg: do we have a list of which widgets should be implemented first? In the reports for the #2155 and M7, I did not find any CREATE\_WIDGET statements.

**#24 - 11/20/2013 05:02 PM - Vadim Gindin**

Some intermediate results:

- 1) Here is three ways creation of dynamic widgets in static containers, creation of dynamic containers (frames and windows) and creation of dynamic widgets in dynamic containers. I'll start from easiest: dynamic widgets for static containers.
- 2) Interesting moment. Appearing properties have not default values and should be set manually. When dynamic widget is being created (appearing properties are set) but its container (frame for example) is not set - widget cannot be shown. Progress shows corresponding error while trying to process appearing properties (SENSITIVE attribute for example). This error may look like this: **\*\*Unable to process SENSITIVE attribute. Button widget is not in a frame.** When I set the frame attribute of a button but didn't set appearing properties - a button won't be shown silently.
- 3) Here are two ways of work with dynamic widgets: assigning properties in same statement (assign phrase in create widget statement) or in separate assign statement. This feature makes the moment of dynamic widget drawing unobvious. From one side I cant draw the widget during CREATE WIDGET processing because of not all properties could be set. From the other side I have to draw dynamic widget without additional statement (like enable or display). So when to do it?
- 4) I think that there should be new method in the ThinClient that combines the logic of pushScreenDefinition and enable methods in the context of dynamic widget been created.
- 5) Call the client from some place.. At this moment I don't know from where.
- 6) I think we should not process dynamic widgets in enable methods - explicitly skip it.

**#25 - 11/20/2013 06:42 PM - Greg Shah**

Greg: do we have a list of which widgets should be implemented first? In the reports for the #2155 and M7, I did not find any CREATE\_WIDGET statements.

For the server project (M7), we only need the CREATE WINDOW support (see note 3 above). It does use the "embedded ASSIGN" too.

**#26 - 11/21/2013 03:57 AM - Constantin Asofiei**

Vadim Gindin wrote:

- 2) ... When I set the frame attribute of a button but didn't set appearing properties - a button won't be shown silently.

But I expect for the i.e. button to be attached to the frame after its FRAME attribute is set, even if is not yet visible.

3) Here are two ways of work with dynamic widgets: assigning properties in same statement (assign phrase in create widget statement) or in separate assign statement. This feature makes the moment of dynamic widget drawing unobvious. From one side I cant draw the widget during CREATE WIDGET processing because of not all properties could be set. From the other side I have to draw dynamic widget without additional statement (like enable or display). So when to do it?

This is logical, a widget will be part of the UI only when is attached to a frame. Also, you need to differentiate between attaching a widget to the frame and displaying/enabling it. An attach will only register the widget with the client-side, while i.e. display/enable will change visibility for an existing, attached widget.

4, 5) Remember, attaching a widget and enabling/disabling it are different things.

6) I think we should not process dynamic widgets in enable methods - explicitly skip it.

Not sure about this. There is the ENABLE ALL case, which enables all widgets in a frame:

```
def var h as handle.  
def var ch as char.  
  
form ch with side-labels size 20 by 10 frame f1.  
  
create fill-in h.  
h:frame = frame f1:handle.  
h:row = 5.  
h:visible = true.  
  
enable all with frame f1.  
  
wait-for go of frame f1.
```

This will enable all the fill-in's in the frame.

Greg: beside for the CREATE WINDOW statement, there are the IN WINDOW clause and the assignment of CURRENT-/DEFAULT-/ACTIVE-WINDOW handles to a dynamic window, which I think should be implemented too, for M7 (or at least checked how they work). As a side note, CREATE WINDOW works only in a GUI environment, in 4GL; the batch mode and the remote appserver requests do not allow it, either. Any idea why this is needed for the server project? Do they run the server in a GUI mode?

**#27 - 11/21/2013 02:03 PM - Greg Shah**

Greg: beside for the CREATE WINDOW statement, there are the IN WINDOW clause and the assignment of CURRENT-/DEFAULT-/ACTIVE-WINDOW handles to a dynamic window, which I think should be implemented too, for M7 (or at least checked how they work).

The IN WINDOW support is included in task [#2027](#).

The CURRENT-/DEFAULT-/ACTIVE-WINDOW support is in [#2018](#).

As a side note, CREATE WINDOW works only in a GUI environment, in 4GL; the batch mode and the remote appserver requests do not allow it, either. Any idea why this is needed for the server project? Do they run the server in a GUI mode?

No, they don't run it in GUI mode. It is a combination of appserver and some batch clients. The customer server code also runs on UNIX (Solaris).

I have assumed that this code works "enough" on the server to allow the core functionality to operate properly OR the code is not used on the server. We do know they have a bunch of GUI code that is mixed into the server code because it is in shared files.

We will certainly need this work for the GUI project anyway.

**#28 - 11/21/2013 04:19 PM - Vadim Gindin**

I will probably repeat the Constantin's question, but I'm recalling, CHUI supports only one (default) window and CREATE WINDOW won't work in CHUI. If we implement CREATE WINDOW in spite of this fact - how to test it?

**#29 - 11/21/2013 04:44 PM - Greg Shah**

CHUI supports only one (default) window and CREATE WINDOW won't work in CHUI. If we implement CREATE WINDOW in spite of this fact - how to test it?

Test what happens in the ChUI/batch mode. Are there errors? If so then throw errors in the same way. Does it partially work or silently fail? We should work the same way. Does it actually work even though the docs say it should not?

**#30 - 11/22/2013 04:55 PM - Vadim Gindin**

Simple test:

```
def var mywin as widget-handle.  
create window mywin.
```

It fails with error message:

```
CREATE WINDOW in character mode is not supported. (4139) "
```

It fails in batch mode (".bpro" command) but with other message:

```
CREATE WINDOW in character-mode is not supported. (4139)  
Cannot create widget of type WINDOW. (3181)
```

**#31 - 11/22/2013 07:08 PM - Greg Shah**

Good. Make sure to generate the proper errors in our code. That means that we should be pretty safe for M7.

**#32 - 11/25/2013 12:24 PM - Vadim Gindin**

How to distinguish batch mode from usual mode?

**#33 - 11/25/2013 12:47 PM - Greg Shah**

Right now we store it on the server in EnvironmentOps. See isBatchMode() and setBatchMode() in that class.

We also have work in [#1787](#) that is going on right now to implement proper batch mode support on the client (for all of the UI features that have different behavior). As part of that, I would expect that we will have knowledge of batch mode on the client too.

**#34 - 11/25/2013 05:04 PM - Vadim Gindin**

- File *widget\_del.p.ast* added

I'm trying to implement conversion of full form of CREATE WIDGET statement - a form with assign statement.  
I wrote simple procedure to solve conversion problems of create widget someHandle assign .. problems. Here it is:

```
def var wh-tmp as widget-handle.
```

```

create button wh-tmp
  assign
    visible = true
    row = 2.

```

I ran conversion process and got intermediate files \*.parser and \*.ast. Here is the \*.parser file:

```

block [BLOCK] @0:0
  statement [STATEMENT] @0:0
    def [DEFINE_VARIABLE] @1:1
      wh-tmp [SYMBOL] @1:9
      as [KW_AS] @1:16
      widget-handle [KW_WID_HAND] @1:19
    statement [STATEMENT] @0:0
      create [CREATE_WIDGET] @3:1
        button [KW_BUTTON] @3:8
        wh-tmp [VAR_HANDLE] @3:15
        assign [KW_ASSIGN] @4:7
          = [ASSIGN] @5:17
            visible [ATTR_LOGICAL] @5:9
            expression [EXPRESSION] @0:0
              true [BOOL_TRUE] @5:19
          = [ASSIGN] @6:13
            row [ATTR_DEC] @6:9
            expression [EXPRESSION] @0:0
              2 [NUM_LITERAL] @6:15

```

It looks correctly. ASSIGN statement is a child of CREATE BUTTON statement.

But let's look at widget\_del.p.ast (attached to this task). Here we can see that ASSIGN statement is not a child but it's a "brother" of CREATE WIDGET statement. I think it's wrong. I don't understand why this happens. I reviewed grammar rules (progress.g), related to this statements. It looks correct: attribute\_assign\_clause is a child of create\_widget\_stmt. Could you advice me why AST incorrectly formed during conversion.

P.S. About implementation. I'm going to make DynamicWidgetFactory.create(..) methods returning created widget to be able to assign the attributes to it after those calls. It may looks like this:

```

..
GenericWidget widget = DynamicWidgetFactory.createButton(..);
widget.setLabel("Some button");
..

```

Am I on the right way?

### #35 - 11/25/2013 05:39 PM - Greg Shah

I am surprised that the "embedded assign" portion is not working. I think it was working properly back in April.

But let's look at widget\_del.p.ast (attached to this task). Here we can see that ASSIGN statement is not a child but it's a "brother" of CREATE WIDGET statement. I think it's wrong. I don't understand why this happens.

This is intentional. Please see rules/annotations/embedded\_attribute\_assign\_rewrite.rules. The idea is that it is supposed to emit exactly as you have described (on the following lines as separate setter calls).

### #36 - 11/26/2013 01:32 AM - Constantin Asofiei

Vadim Gindin wrote:

I'm trying to implement conversion of full form of CREATE WIDGET statement - a form with assign statement.

The conversion part should be working properly. Your test converts to this:

```
DynamicWidgetFactory.createButton(whTmp);  
handle.unwrapWidget(whTmp).setVisible(new logical(true));  
handle.unwrapWidget(whTmp).setRow(new integer(2));
```

Considering that a new dynamic resource (widget or not) is ALWAYS saved in a handle, and that handle is used in any subsequent assign or other statement, I don't see where the problem is.

### #37 - 11/26/2013 11:50 AM - Vadim Gindin

I'm sorry, I muddled files with same names in different folders.. My mistake. You're right. Embedded assign works correctly. The problem is in a triggers phrase. The same procedure with triggers:

```
def var wh-tmp as widget-handle.
```

```
create button wh-tmp  
  assign  
    visible = true  
    row = 2  
  triggers:  
    on choose  
      message "Choosed".  
  end.
```

Converted code contains compile error:

```
DynamicWidgetFactory.createButton(whTmp, "choose", TriggerBlock0.class, WidgetDel.this);
handle.unwrapWidget(whTmp).setVisible(new logical(true));
handle.unwrapWidget(whTmp).setRow(new integer(2));
```

There is no createButton method with this signature. I'm digging in it.

#### #38 - 11/26/2013 11:54 AM - Constantin Asofiei

AFAIK the trigger phrase in the CREATE statement is not used by the server project, so should not be worked by [#2027](#). Please focus on adding the CHUI/batch-mode errors for CREATE WINDOW and check how IN WINDOW clause works in CHUI/batch mode.

#### #39 - 11/26/2013 04:15 PM - Vadim Gindin

- File *vig\_upd20131127a.zip* added

I'm posting update about CREATE WINDOW errors.

IN WINDOW clause is used in a big amount of statements:

ENABLE, frame phrase, MESSAGE, HIDE, PAUSE, PROMPT-FOR, STATUS, SYSTEM-DIALOGs, VIEW, DISPLAY. In all cases if IN WINDOW clause is omitted current window is used (documentation says). Should I test all of these statements?

I tried to test MESSAGE and DISPLAY statements:

```
message "text" in window default-window:handle.
```

It works fine in CHUI mode and don't work in batch mode, showing the error:

```
Cannot access the HANDLE attribute because the widget does not exists. (3140)
Could not find widget handle for window from expression. (3138)
```

DISPLAY, ENABLE statements shows the same result: work in CHUI mode and show this error in batch mode.

Should I test all statements?

I also converted first test to java. Here is the result:

```
message("text", defaultWindow());
```

It shows compile error because message(..) methods doesn't support window parameter. Should I make support of this clause for all statements in this task?

The server project uses the IN WINDOW clause only with the DISPLAY statement, in only one program; that program acts like this:

- determines which window to use, depending on some conditions:
  1. if not batch mode, then: if some condition, then use current-window, else use a dynamically created window
  2. if batch mode, use default-window handle.
- uses a DISPLAY ... IN WINDOW h. to output some data.

The code snippet would look like this:

```
def var wh as handle.

if not session:batch-mode
then do:
  if <some-condition> then do:
    output to some-file.txt.
    wh = current-window.
  end.
  else create window wh assign width = 180.
end.
else wh = default-window.

display <data> with frame ff in window wh.
```

And something else: although an error is displayed, the CREATE WINDOW does initialize the handle to something, in CHUI mode:

```
def var h as handle.
create window h assign width = 180. /* this will show error */
message h h:width current-window default-window active-window. /* all handles are the same here */
```

The good news is that all current-/default-/active-window handles refer the same resource and also in CHUI mode CREATE WINDOW will assign the handle to the same window. More, in chui mode, changing the WIDTH is a no-op.

About the update:

1. your sources are a little old, please make sure you work on the latest revision
2. DynamicWidgetFactory: do not add constants with error message text; their usage is limited only to certain methods (i.e. where the error is displayed), so there is no gain from this.
3. I guess you didn't check how the ERROR-STATUS is affected by this statement:

```
def var h as handle.

create window h no-error.

message error-status:error error-status:num-messages error-status:get-message(1)
error-status:get-message(2) .
```

In chui mode, this shows that a condition is not raised and the error-status:error flag is not set, but error messages are recorded; use a `ErrorHandler.recordOrShowError(<error-codes>, <error-texts>, false, false, false)` for this case.

In batch mode, this shows that the error-status:error flag is set and error messages are recorded; please use a `ErrorHandler.recordOrThrowError(<error-codes>, <error-text>, false, false)` for this case. Also, do not use multiple ErrorHandler API calls, when more than one error message is recorded/thrown - do this in one call. To test that a condition is raised in batch mode, use this:

```
def var h as handle.
output to some-file.txt.
do on error undo, leave:
  create window h.
  message "this must not be reached".
end.
output close.
```

To conclude, for [#2027](#):

1. make sure you have tests which check the error for both modes
2. in chui mode, allow CREATE WINDOW to pass and initialize the handle with LogicalTerminal.currentWindow
3. make sure the DISPLAY ... IN WINDOW statement works for both valid and invalid handles

All other findings (default-/current-/active-window behaviour in batch/chui mode, width attribute, etc) can be deferred for the [#2018](#) task.

#### #41 - 11/29/2013 02:55 PM - Vadim Gindin

- File vig\_upd20131130a.zip added

Well, I'm close to finish this task. See next update.

By the way DISPLAY methods are not implemented. Will we make it in other task?

#### #42 - 11/30/2013 08:54 AM - Constantin Asofiei

Update review:

1. DynamicWidgetFactory.createWindow - I think you are missing a return, in case the window can't be created and should default to current-window... More, if in batch mode and NO-ERROR clause is present, the h.assign(LogicalTerminal.currentWindow()); code will still execute, when it shouldn't:

```
public static void createWindow(handle h, character widgetPool)
{
    if (!isAbleToCreateWindow())
    {
        h.assign(LogicalTerminal.currentWindow()); /* this needs to be executed only if non-batch mode. */
    }
    // a return is missing here
    return;
}

create(LegacyResource.WINDOW, h, widgetPool);
}
```

2. LogicalTerminal.applyChanges contains changes not related to this task.
3. the conversion changes I guess are trying to fix a DISPLAY ... IN WINDOW statement without a frame-phrase?

By the way DISPLAY methods are not implemented. Will we make it in other task?

No, add it now; for CHUI/batch mode, the DISPLAY ... IN WINDOW will either:

- perform the DISPLAY, if the passed handle is a valid handle and is a window
- show some error if the passed handle is invalid or is not a window.

GenericFrame.display(..., handle hWin) must:

1. validate the passed handle and show proper error messages if the handle is invalid or not a window.
2. if the handle is valid and a window, go ahead and call the GenericFrame.display APIs which don't take a window handle (as in CHUI mode this will always target current-window).

- File vig\_upd20131204a.zip added

My next update.

Constantin Asofiei wrote:

Update review:

1. DynamicWidgetFactory.createWindow - I think you are missing a return, in case the window can't be created and should default to current-window... More, if in batch mode and NO-ERROR clause is present, the h.assign(LogicalTerminal.currentWindow()); code will still execute, when it shouldn't:

ok

1. LogicalTerminal.applyChanges contains changes not related to this task.

Yes, this class contains changes for two tasks. I will clear it close to regression testing if you don't mind.

1. the conversion changes I guess are trying to fix a DISPLAY ... IN WINDOW statement without a frame-phrase?

It works with and without frame-phrase.

By the way DISPLAY methods are not implemented. Will we make it in other task?

No, add it now; for CHUI/batch mode, the DISPLAY ... IN WINDOW will either:

- perform the DISPLAY, if the passed handle is a valid handle and is a window
- show some error if the passed handle is invalid or is not a window.

GenericFrame.display(..., handle hWin) must:

1. validate the passed handle and show proper error messages if the handle is invalid or not a window.
2. if the handle is valid and a window, go ahead and call the GenericFrame.display APIs which don't take a window handle (as in CHUI mode this will always target current-window).

I've implemented.

#### #44 - 12/05/2013 09:52 AM - Constantin Asofiei

DynamicWidgetFactory:

1. max line width is 98, so please format accordingly
2. something I forgot to mention: widget pool validation for CREATE statements is done before anything else; please add tests using the IN WIDGET-POOL clause (the named pool name either unknown or for an not-existing existing pool) and make sure it works properly.
3. should have mentioned it earlier, but the CREATE WINDOW can have a CREATE VALUE(ch) version, where ch might (or might not be) a window string. Thus, move the window-related code to the create(String widgetType, handle h, character widgetPool) method, AFTER the widget-pool and the widget-type validation (and executed only if a WINDOW is supposed to be created).
4. I don't understand this test in createWindow:

```
if (EnvironmentOps.isBatchMode().booleanValue() && ErrorManager.isSilent())
{
    h.assign(LogicalTerminal.currentWindow());
}
```

In note 42, I mentioned that handle assignment "needs to be executed only if non-batch mode". Your test is checking if in batch mode; did you find otherwise?

5. Don't understand the need of the ErrorManager.isSilent() check: please explain why you use it; if NO-ERROR clause is used, then any error is silently recorded (no condition is thrown).

GenericFrame:

1. max line width is 98, so please format accordingly
2. the comment for the error message in isAbleToDisplayWindow:

```
// TODO define resource name (BUTTON, BUFFER, ..) and write in ()
String ms = "Wrong type of object handle () "
    + "-- must be a window handle";
ErrorManager.recordOrThrowError(3139, ms);
```

I think it relates to the fact that the handle's resource type is used in the error message. For this, do a handle.unwrapType(hWin).getResourceType() to obtain the resource type.

3. internally, you can use the handle.\_isValid API which returns a Java boolean to avoid using the handle.isValid which returns a 4GL-style logical.

LogicalTerminal:

1. the file needs to be merged.

**#45 - 12/06/2013 06:14 PM - Vadim Gindin**

- File *vig\_upd20131206a.zip* added

Here is the widget-pool (in CREATE statements) simple tests:

```
def var mywin as widget-handle.  
create widget-pool "p".  
create window mywin in widget-pool "p".
```

Works as without pool: shows an error CREATE WINDOW in character mode is not supported (4139)

```
def var mywin as widget-handle.  
/*create widget-pool "p".*/  
create window mywin in widget-pool "p".
```

Shows an error Trying to create a widget in widget-pool "p", but pool does not exist. (3178).

```
def var mywin as widget-handle.  
def var p as char.  
p = ?.  
create window mywin in widget-pool p.
```

Shows an error Unable to evaluate widget pool name. (3177)

These tests works as expected (after I moved window-related code to create(...) method as you advised).

All other remarks are corrected. See next update.

P.S. You are right about isSilent() method. My mistake - it is really not needed there. I removed it.

**#46 - 12/07/2013 05:37 AM - Constantin Asofiei**

OK, the implementation looks good. Please merge with the latest revision, remove any changes in LogicalTerminal that are not related to this task and get it regression tested.

This work for will be enough for the server project. Greg: should we continue with [#2027](#) further on, with the other dynamic widgets?

**#47 - 12/09/2013 06:16 PM - Greg Shah**

We will come back to the rest of this later. Let's get the M7 changes tested and committed.

**#48 - 12/18/2013 01:02 PM - Vadim Gindin**

Regression passed, committed to bzt rev #10421.

**#49 - 12/18/2013 01:07 PM - Greg Shah**

Please post the final changes.

**#50 - 12/18/2013 01:11 PM - Vadim Gindin**

- File *vig\_upd20131217a.zip* added

Here it is.

**#51 - 06/16/2014 04:44 PM - Greg Shah**

One issue that needs to be addressed is how the order of attribute assignment (in the CREATE widget statement) can affect the success/failure of actual assignment that occur at runtime. Some examples can be seen in [#1789](#) note 30. In some cases an error message may be displayed. In other cases, the attribute assignment may be silently bypassed.

In P2J we may just generate an NPE (which is not correct). Considering that silent bypassing of assignments can be present in working applications, we must make sure P2J operates the same way as the 4GL in this regard.

**#52 - 06/21/2014 05:07 AM - Greg Shah**

The window-related features will be implemented in a different task. For [#2028](#), focus on the general dynamic widget support like getting the attributes assigned properly, handling error processing properly, getting deletion supported properly. If you avoid the CREATE WINDOW right now, then you can test this in ChUI and in P2J.

**#53 - 07/01/2014 09:01 AM - Eugenie Lyzenko**

Some general considerations.

The first issue discovered for dynamic widgets is related to the proper attribute setting. The key problem is when dynamic widget is creating - the widget id is not being set. And P2J code to set the attribute value uses widget id to get the particular widget instance. For id == -1 the widget instance is null and this causes the NullPointerException.

The proper is assigned to the static widget when the frame containing the widget is initialized(`GenericFrame.CreateFrame()`, `GenericFrame.SetupFrame()` - the method `GenericFrame.coreInitialize()`). For dynamic widgets this procedure is now omitting. The P2J code makes binding between widgets and the frame that owning the widgets. Every widget id is tightly related to the frame id it belongs.

The dynamic widgets has no such dependency. The good place to assign the correct widget id for dynamic widget is the moment when we assign the FRAME attribute of the dynamic widget. On the other hand we need to implement possibility to work with dynamic widgets without/before FRAME attribute initialized(as it is happening in 4GL). Another word we need some container that holds the dynamic widget and allows to dereference/manipulate with widget internals until it is assigned to some frame. After this assignment all widget operations will become up to the frame that become the owner of the dynamic widget.

#### #54 - 07/02/2014 05:03 AM - Eugenie Lyzenko

Additional investigations show the 4GL still has limitation for usage attribute setting and corrected FRAME attribute value. Some attributes like SENSITIVE requires the FRAME attribute to be set properly or before the requesting for SENSITIVE to be set. Otherwise the error message is displaying:

```
...
**Unable to process SENSITIVE attribute.  TOGGLE-BOX widget is not in a frame.
(4073)
Press space bar to continue.
...
```

or

```
...
**Unable to process SENSITIVE attribute.  BUTTON widget is not in a frame.
(4073)
Press space bar to continue.
...
```

The application continues execution declining the action that produces the error. The other attribute setting(like ROW or COLUMN or even LABEL) does not generate such error.

The other types of error like setting coordinates that causes the widget or part of it to be outside the containing frame is critical and after displaying error like this:

```
...
TOGGLE-BOX widget does not fit in parent FRAME f1. (4041)

Procedure complete. Press space bar to continue.
...
```

stops the application execution.

**#55 - 07/02/2014 07:01 AM - Greg Shah**

The good place to assign the correct widget id for dynamic widget is the moment when we assign the FRAME attribute of the dynamic widget.

I think this is too late for dynamic widgets.

On the other hand we need to implement possibility to work with dynamic widgets without/before FRAME attribute initialized(as it is happening in 4GL).

Yes, this is exactly why we must not have the widget ID depend on setting the FRAME attribute. Before we decide on an approach, please make a list of the places in our code where we are dependent upon the relationship between a widget ID and its containing frame's ID. For example, on the server side there is `GenericFrame.getWidgetForId()` and on the client side there is use of `WidgetId.frameIdFromWidgetId()` in `ThinClient.view_()`. I know there are many other locations with this dependency.

It doesn't seem like our current approach is a good match for the 4GL. It seems like the 4GL just assigns IDs sequentially. Currently, I don't know of a reason why we must match their algorithm exactly, but it may make sense to be closer to their implementation, especially if there are other good reasons for it.

The application continues execution declining the action that produces the error. The other attribute setting(like ROW or COLUMN or even LABEL) does not generate such error.

We need to plan for how to support this behavior in a clean way. I don't want every method to be duplicating lots of error handling, but we need to be able to match the runtime error processing where needed.

**#56 - 07/03/2014 07:03 PM - Eugenie Lyzenko**

I think the following methods are involved in widget-frame ID relationship in more or less degree:

```
server side processing
-----
```

```
GenericWidget:
isEntered() <- GenericFrame.getFrameBufferRef().isEntered(config.getID())
```

```

isNotEntered() <- GenericFrame.getFrameBufferRef().isEntered(config.getID())
apply() <- LogicalTerminal.apply(GenericFrame frame, int widgetId, String event)
setFormat() <- LogicalTerminal.setFormat(int widget, String format)
pushScreenDefinition() <- GenericFrame.pushScreenDefinition()

GenericFrame:
importSharedFrame()
pushScreenDefinition() - prepare the new screen definition based upon frame-widget ID's relationship
getSelection() <- LogicalTerminal.getSelection(int widgetId)
underline() <- LogicalTerminal.underline(GenericFrame frame, int[] widgetIds)
getWidgetForId(int id)
wrapWidgetsToFieldGroup() - uses widget array to prepare the field group
view(FrameElement[] data) - ID based frame content viewer
setWorker()
coreInitialize()
updateDynamicBrowseTitle()
makeFrameEnable()
makeFrameVisible()
idsFromWidgets()
widgetListFromFrameElements()

ScreenBuffer:
getState()
setState()
getValueWidgetId()
setValueWidgetId()
isEntered()
resetEntered()
mergeChanged(ScreenBuffer diff)
getChanged()
getBaseId()

LogicalTerminal:
setLocation() <- ThinClient.setLocation()
refreshFrameWidget() <- ThinClient.refreshFrameWidget()
waitFor() <- ThinClient.waitFor()
getCurrentFrame() <- ThinClient.identifyFrame()
enable()/disable()/view()
getAnyWidgetForId() <- ThinClient.getAnyWidgetForIdInt()
(get/set)firstTabItem()
getNextTabItem()

WidgetId
Full class

The frame should contain the valid widgets[] array inside.

client side processing
-----
ThinClient:
getFrame()
getWidget()
getEnablePendingFocus()
enableWorker()
processEnable() <- Frame.enabledWidgetsByList()
nextPrompt() <- Frame.setFocusId()
view_() <- WidgetId.frameIdFromWidgetId()
refreshFrameWidget() <- WidgetRegistry class
setScreenBuffer()

Frame:
getFocusId()
enabledWidgetsByList()
setFocusId()

WidgetRegistry:
addWidgetsToFrame()

UiUtils:
lookupWidgetId()

```

As general considerations: Currently we encode the frame ID in every widget ID by assigning the number 1000, e.g. 1001 means first widget in first frame, 2014 - 14 widget in 2 frame. This is reasonable and looks useful to detect static frame-widget relationship. I would suggest to keep this approach but extending it: the higher octet of the 32-bit int value to give for frame ID, lower - for widget ID. This gives us 65535 frames and 65535

widgets for each frame for static widgets. For dynamic widgets the frame half of ID will be 0 meaning there is no frame binding at this time, limiting the total number of possible dynamic widgets to 65535. And we will need the single place method to calculate/manage the widget ID for create/delete for dynamic widgets to avoid multiple inline ID calculation.

#### #57 - 07/04/2014 08:32 PM - Eugenie Lyzenko

Another consideration for dynamic widget ID. According to Progress documents the dynamic widgets exist until explicit deletion or session ends. So the widget number for dynamic case should be global per session. On the other hand if we incorporate the frame ID into widget ID when frame attribute is assigned the resulting dynamic widget ID should not accidentally duplicate already existed ID for static widget in the same frame. So in the widget ID we need to have the flag field indicating whether the widget is static or dynamic. This can be the highest bit of the 32-bit integer for example or something like this. In this model we will have 65535 widgets for every frame and 65535 global dynamic widgets for any frame to be attached to.

#### #58 - 07/25/2014 06:19 PM - Eugenie Lyzenko

Some new investigation results. The widgets are adding while frame creation.

```
GenericFrame.createFrame()
{
    The widget is taken from the WidgetList map.
    Then for every widget in a frame the frame attribute is initialized and the Id attribute is calculated and set.
    After this the new screen buffer is creating for the frame and its content with UNINITIALIZED state.

    Then GenericFrame.pushScreenDefinition() adds frame config and widgets configs to the freshly created ScreenDefinition instance.
    Then LogicalTerminal.pushScreenDefinition() is called causing the client side to transfer the frame content from server to the client.
}
```

At this moment every widget(including frame itself) has valid id to reference/enumerate the widget object(from ScreenDefinition widgets[] array) and associated frame on client side and this is persistent during the frame life cycle.

Client side retrieve the widget from WidgetRegistry class by integer id. This registry is filling on the frame/widget reconstruction stage.

The idea of the dynamic widget implementation can be based on the fact the widget id is not directly related to the frame id it belongs, meaning the id for widget can be more than frame id + 999. The only requirement for widget id is it must be unique. So we can set up some id generator inside DynamicWidgetFactory, something like:

```
...
/** The base for new Id to be set for new widget creation. */
// This is the just starting point, can be other to not intersect with static widget id values
private static final int DYN_ID_BASE = 32768;

/** Keeping track of the next Id number to assign for widget creation. */
private static int nextIdAvailable = DYN_ID_BASE + 1;
...
/**
 * Retrieve the unique per session integer id number for next new widget to create.
 *
 * @return The integer Id number for next dynamic widget to create.
 */
public static int getIdForNewWidget()
{
    return nextIdAvailable++;
}
...
```

This method will return new id for every new widget creation:

```
...
```

```

    public static void create(String widgetType, handle h, character widgetPool)
    {
...
        try
        {
            Constructor<? extends GenericWidget> ctor = widCls.getDeclaredConstructor(boolean.class);
            WrappedResource res = ctor.newInstance(true);

            h.assign(res);

            int newId = getIdForNewWidget();
            h.unwrapWidget().setId(newId);

            WidgetPool.addResource(res, widgetPool);
        }
...

```

Then the next key point in dynamic widget life cycle is the assigning the frame attribute. For dynamic widget we can modify the respective method in following way(GenericWidget.java):

```

...
public void setFrame(GenericFrame frame)
{
    this.frame = frame;

    // For dynamic widgets this is the place where we need to initialize all frame internals
    // related to the widget integration into the frame functionality
    if (dynamic)
    {
        frame.addDynamicWidget(this);
    }
}
...

```

in the GenericFrame.addDynamicWidget(GenericWidget) method we can concentrate all required calls to make new dynamic widget operational, like this(GenericFrame.java):

This is just schematic implementation to demonstrate general approach

```

...
/**
 * Adding new dynamic widget to the current frame.
 *
 * @param dynWidget The widget to be added to current frame.
 */
void addDynamicWidget(GenericWidget dynWidget)
{
    n2w.put("dynWidget", dynWidget); // The name certainly should be something more real and unique

    // Add new widget to the widgets array

    // This implementation should be replaced with dynamic size array for efficiency
    GenericWidget[] newWidgets = new GenericWidget[n2w.size()];
    for (int i = 0; i < widgets.length; i++)
    {
        newWidgets[i] = widgets[i];
    }
    newWidgets[widgets.length] = dynWidget;
    widgets = newWidgets;

    // Refresh frame definition if exists(if not to do this the client will not see the new widget)
    if (frameDef != null)
    {
        frameDef.addConfig(dynWidget.getConfig());
    }
}
...

```

This works in simple test and I think this approach is potentially implementable.

#59 - 07/28/2014 01:28 PM - Greg Shah

Sorry it has taken me so long to review your results.

As general considerations: Currently we encode the frame ID in every widget ID by assigning the number 1000, e.g. 1001 means first widget in first frame, 2014 - 14 widget in 2 frame. This is reasonable and looks useful to detect static frame-widget relationship.

Unfortunately, this is no longer appropriate. It does not match how Progress assigned IDs. Progress seems to have a very simple ID algorithm, which just keeps track of the next ID number available, when it assigns a new ID, it seems to just increment that integer. It doesn't leave gaps and it doesn't encode special information (like the containing frame) in the ID.

We are going to move to an equivalent system. Encoding the frame ID into the widget ID was a short-cut that is now doing more harm than good.

I would suggest to keep this approach but extending it: the higher octet of the 32-bit int value to give for frame ID, lower - for widget ID. This gives us 65535 frames and 65535 widgets for each frame for static widgets. For dynamic widgets the frame half of ID will be 0 meaning there is no frame binding at this time, limiting the total number of possible dynamic widgets to 65535.

I don't want to do this, because it just makes things even less compatible with Progress. I am not requiring us to generate IDs the exactly same way as Progress, but I want it to be very close. The new dynamic widget support is the core reason to simplify things right now.

Frame IDs, window IDs and widget IDs will all come from the same exact "nextWidgetId" integer.

And we will need the single place method to calculate/manage the widget ID for create/delete for dynamic widgets to avoid multiple inline ID calculation.

I agree that the nextWidgetId will be stored once per-session and it will be used from all places that create a new widget (whether from a frame definition or from a dynamic CREATE WIDGET statement).

The only requirement for widget id is it must be unique. So we can set up some id generator inside DynamicWidgetFactory, something like:

I don't want to have separate ID "address space" for dynamic and static widgets. Progress does not do that. We won't do that.

Instead, we can just assign the ID as soon as a widget is instantiated (no matter whether it is dynamic or not). Then it will be there already, whenever the frame or other code needs it.

This means we need to know:

1. All the places in the code that must be fixed because they rely upon the frame ID to be calculated from a widget ID. These places can lookup the frame from the widget, since the containing frame is something that the widget should be able to report. In other words, we are removing the "short-cut" lookup approach. Of course, some simple helpers can be added to our widget code to make this easy to call.
2. All the places that assign new widget IDs (to a frame, widget or window) that must be changed to use the new simple algorithm for assignment. The WidgetId class and helpers in LogicalTerminal (e.g. findNextID()) are where we do this today. I'd like to hide this all inside the WidgetId class in the changed version (there is no reason for LogicalTerminal to have this code).

The idea of the dynamic widget implementation can be based on the fact the widget id is not directly related to the frame id it belongs, meaning the id for widget can be more than frame id + 999.

Not in the new approach.

Please focus on implementing ONLY the new ID approach for now. Let's get this working, regression tested and checked in first. Then you can followup with a more complete dynamic approach.

**#60 - 07/31/2014 05:32 PM - Eugenie Lyzenko**

- File `evl_upd20140731a.zip` added

This update for your review is the starting point for new widget approach. The public method in `WidgetId` class:

```
getIdForNewWidget() to request the ID for new widget  
freeIdToUse() to signal the given ID is no more using and can be distributed to another new widget.
```

The internal idea of implementation is to have some ID pool from where we can get next available ID for new widget. The pool is 0-based `ArrayDeque`. Initially we allocate some amount of the ID values to return to new widget. If the pool become exhausted - we allocate next portion of ID to distribute. If at run time the ID is no more used(widget become deleted) - the ID returns to the pool to be used further. This avoids ID leakage when multiple widget delete/create cycles encountered in a code.

Continue working. The next step will be to find all ID usage for static widgets and shifts to the new simple approach isolating the ID internals inside the `WidgetId` class.

**#61 - 08/01/2014 03:02 PM - Greg Shah**

Code Review 0731a

We can't use this approach.

First of all, widget ID allocation will be used on the server. The server CANNOT use static data to store IDs because it must be per-session not JVM-wide. Context local data must be used.

From what I have seen, the Progress widget ID allocation approach is a simple sequential increment. Can you find any scenario where the 4GL doesn't do this? They may have some protection for never re-allocating the same ID, but if they don't re-use IDs then only in a wrapping case could they re-use something. Who knows if Progress even checks such things? Before we implement an approach that recycles widget IDs, please create some Progress 4GL testcases to try to determine how this works. Some questions:

1. What is the starting point for the first ID ever created for the session?
2. When you allocate a large number (1000 or more) of IDs by using a `CREATE WIDGET` statement) and then you deallocate some (10 or 100) of those IDs, does Progress ever re-use those IDs?
3. If you `CREATE` and `DELETE` widgets for a very long time (millions or billions), can you get the widget ID to "wrap around" and become negative or go back to its starting point?

Please note that you can't just use `CREATE` without ever using `DELETE` because you would run out of memory.

Also note that you should not run your tests from the procedure editor, but instead do it from the command line with the `-p <testcase_name>` option. Otherwise your results might be invalid because the procedure editor is itself 4GL code.

**#62 - 08/01/2014 03:48 PM - Eugenie Lyzenko**

From what I have seen, the Progress widget ID allocation approach is a simple sequential increment. Can you find any scenario where the 4GL doesn't do this?

Consider the case when after some static widget allocations Progress creates 10 dynamic widget, then say 2 of them are freed. And then creates the dynamic widget again. If the old ID are not used - the new number will be assigned leaving gaps in widget ID "array". In this scenario it is possible to exceed the ID capacity (for example 32-bit integer) having only several widgets that will be created and deleted.

And the question. What is the widget ID in 4GL, is it attribute or something else? Another word what is the best way to check the widget ID in 4GL? Or this is undocumented internal feature we can not explicitly check?

**#63 - 08/01/2014 03:56 PM - Eugenie Lyzenko**

The only I've found is WIDGET-ID attribute. But it is declared as working only in Windows GUI. Is it what we implementing here?

**#64 - 08/01/2014 04:02 PM - Greg Shah**

We can at least use it for now to see how Progress creates its widget ids. Yes, it only works on GUI and without it the default value is reported as ?.

We are not necessarily trying to duplicate this feature exactly, right now. But on the other hand, I would like to be more compatible rather than less.

Please determine how it works and then we can decide on an approach.

**#65 - 08/04/2014 02:17 PM - Eugenie Lyzenko**

- File testcases\_20140804a.zip added

The investigation findings:

1. The starting point for ID allocation is 65535. Neither 0 nor 1. And more important this is the topmost limit. All next ID allocations will have decreased ID, from 65535 to 1.
2. Reusing is not possible. If there are the free ID upper than currently deleted - they will not be used. The next ID == (last allocated - 1) is used instead.
3. The wrapping is working this way. The last possible widget ID is 1. If this number was used the system display the message:

```
**Unable to realize BUTTON widget. (4025)
```

and the widget is not creating.

But then on the further creation will use the next available ID number from the top. It is actually the first available ID not used by static widgets. If there are the dynamic widgets allocated at this time - they will be destroyed. So we returning to the state when no dynamic widgets allocated.

This is the picture of the found implementation. Looks like there is a limit of 65535 (16-bit integer) for dynamic widget creation. The question is will we have to implement this approach for compatibility?

- 16-bit up limit
- decreased allocation

- no re-usage
  - forcing destroy on ID number wrapping
- All features or may be only part of them?

**#66 - 08/04/2014 03:39 PM - Greg Shah**

If there are the dynamic widgets allocated at this time - they will be destroyed. So we returning to the state when no dynamic widgets allocated.

How did you determine that this was the behavior?

**#67 - 08/04/2014 04:39 PM - Eugenie Lyzenko**

How did you determine that this was the behavior?

Consider the testcase dyn\_wid\_test6.p. It has two pair of buttons, one to create/destroy single widget, other to create/destroy 10 widgets simultaneously. The created widgets can exists any time until explicit destroy. It also has buttons to consume 10000, 1000, 100 and 10 widgets in one pressing by create/destroy in a loop.

Initially I made this test to create group of widgets in a some ID distance from start to occupy continuous region of ID and see what is happening when wrap around is reaching. I wanted to see what will be the next ID after 1 when some numbers consumed. Whether the Progress starts from static ID end or after region occupied by existed dynamic widgets.

So when I got 1 number the next creation shows the error message and existed widgets disappears. Trying to explicitly delete them produces the invalid handle error. And ID assigning starts from the end of static widget ID. That's why I concluded the dynamic widgets were terminated.

**#68 - 08/06/2014 05:09 PM - Eugenie Lyzenko**

Another issue we need to solve in P2J implementation is the new implementation for `WidgetId.frameIdFromWidgetId(int widgetId)`. The old one based on the fact the frame id is embedded into the widgetId which will not the case in new implementation(not depending what approach we choose to take). When frame id can be easily computed from widgetId the method is safe to be called either from server or client context(LogicalTerminal, EventDefinition or ThinClient classes). Yes, it is possible to extract frame ID from widget not having frame ID embedding into widget ID because after frame attribute is assigned every widget has frame information inside. But now we need to differentiate the server and client context for this calculation because classes used in this process will be different for server and client sides.

So when I got 1 number the next creation shows the error message and existed widgets disappears.

This is not exactly what is happening. After the message is displayed, I the 4GL raises a STOP condition. If you change the code to have this at the end:

```
DO ON STOP UNDO, LEAVE:  
    ENABLE ALL WITH FRAME f1.  
    WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.  
    MESSAGE "AFTER WAIT-FOR".  
END.  
MESSAGE "AFTER DO BLOCK".
```

You will see the AFTER DO BLOCK message just after the "error" message and then the program will exit.

Consider that the default behavior for a STOP condition is to restart the application (which will just re-run the startup procedure which is dyn\_wid\_test.6 in this case). That makes it look like everything is deleted, but it really isn't. I think you need to rework your tests to take this STOP condition into account and see if the dynamic widgets are really still there.

But now we need to differentiate the server and client context for this calculation because classes used in this process will be different for server and client sides.

This lookup doesn't need to be in the WidgetId class. The client widget classes can have helpers to access their containing frame widget/frame id and the server widgets classes will also have such helpers.

**#71 - 08/07/2014 10:41 AM - Eugenie Lyzenko**

Yes, you are right, just after attempt to create new widget in "wrap around" case - when widget with ID = 1 was just deleted the already existed widgets are not terminating but remains accessible. But the "wrap around" case generates STOP condition exception which reloads the application so all previously created widgets(what is happening after the application is up and running) will be destroyed anyway and the program start executing like after initial start. So:

1. The dynamic widgets live until application stops.
2. It is not possible to pass through ID = 1 down limit without STOP condition. So the ID number is never reused without STOP.

This means Progress has serious constraint for using dynamic widgets and often delete/create is not recommended, correct? And do we really need to implement this behavior?

**#72 - 08/07/2014 11:55 AM - Greg Shah**

This means Progress has serious constraint for using dynamic widgets

Yes.

often delete/create is not recommended

It looks so.

And do we really need to implement this behavior?

Probably so. Before I say for sure, please do this test: run the same test without the -usewidgetid and see if it behaves the same way. The WIDGET-ID attribute will report as ? but otherwise I think it will work.

**#73 - 08/07/2014 01:21 PM - Eugenie Lyzenko**

Probably so. Before I say for sure, please do this test: run the same test without the -usewidgetid and see if it behaves the same way. The WIDGET-ID attribute will report as ? but otherwise I think it will work.

Absolutely different result. There is no STOP condition when expected. It is possible to continue create/delete dynamic widgets after planned "wrap around" place. May be the upper limit is more in this case? 32-bit integer?

**#74 - 08/07/2014 03:03 PM - Greg Shah**

Interesting. We won't implement this feature right now.

Since we can't "see" the widget IDs that are generated in this case, we just need to know if there are any obvious limits that need to be implemented. Please try to CREATE a large number with NO DELETE. Please determine how many can be created before there is some kind of failure. If it is a 32-bit integer, then out of memory may be the limiting factor.

**#75 - 08/08/2014 05:46 AM - Eugenie Lyzenko**

The process of ID saturation has some notes. I made simple creation loop without delete. The results vary from run to run. The maximum I've got is 9972 creations. This looks not adequate. With DELETE following by creation - much more, about several hundred. But I think this can be application design issue, not Progress limitation.

So I created and run another loop with PAUSE 1 NO-MESSAGE after creation. I think it can give us the clear picture. But it takes a time(because 1 second is the minimal pause possible in Progress). But I hope we will finally have the result.

**#76 - 08/08/2014 06:15 AM - Greg Shah**

I made simple creation loop without delete. The results vary from run to run. The maximum I've got is 9972 creations.

Make sure to run this directly from the command line (with -p) instead of from the procedure editor. The procedure editor is running in the same Progress environment and it will take up resources that could cause your results to vary. Otherwise, I don't know why it would vary from run to run.

With DELETE following by creation - much more, about several hundred.

Do you mean "several hundred thousand"?

So I created and run another loop with PAUSE 1 NO-MESSAGE after creation. I think it can give us the clear picture.

What is your thinking behind this? Why do you think the problem is timing related?

**#77 - 08/08/2014 06:36 AM - Constantin Asofiei**

Eugenie, check if this limit is not somehow related to some pro arguments. At least for recursivity issues, I've found that the following enhances the stack depth:

```
-nb 20000 -s 10000000 -mmax 65534
```

There might be other arguments which can affect the available memory allocated for dynamic resources.

**#78 - 08/08/2014 07:11 AM - Eugenie Lyzenko**

Make sure to run this directly from the command line (with -p) instead of from the procedure editor. The procedure editor is running in the same Progress environment and it will take up resources that could cause your results to vary. Otherwise, I don't know why it would vary from run to run.

OK. I'm running the tests with command processor(with or without -usewidgetid):

```
prowin32 -p test.p
```

Do you mean "several hundred thousand"?

Yes, exactly, sorry.

So I created and run another loop with PAUSE 1 NO-MESSAGE after creation. I think it can give us the clear picture.

What is your thinking behind this? Why do you think the problem is timing related?

Without pausing the application become not responsive from Windows point of view and just freezes.

## #79 - 08/08/2014 09:18 AM - Greg Shah

I am interested in the results of your PAUSE version. But we already have enough knowledge to know that there are more than 64K of widgets that can be created in a normal environment.

We know this:

1. Unless the `-usewidgetid` feature is enabled in the 4GL, the widget IDs are not visible directly to 4GL code. We don't have a current need to implement `-usewidgetid` mode right now. Since the widget IDs are not visible, there are no special requirements of what numbers are assigned.
2. The size of the "address space" of widget IDs is unclear right now, but it is large enough that we don't have to limit it to a 16-bit integer.

With this, we can keep a very simple widget ID approach.

- Keep an int counter called `nextId`.
- Let's assume that the DEFAULT-WINDOW will have an ID of 1.
- The initial value of `nextId` can be 2 which can be the ID of the first allocated widget other than the DEFAULT-WINDOW. This should be defined as the constant `WIDGET_ID_RANGE_BEGIN`.
- Keep a boolean flag called `wrapped` which is initialized to false.
- I don't want to keep a list of de-allocated IDs. This just takes up memory, additional processing time and requires notification when widgets are destroyed.
- When a new ID needs to be allocated, the code can look something like this:

```
public int allocateId()
{
    while (true)
    {
        // we are deliberately assigning the value before incrementing
        int candidate = nextId++;

        if (wrapped)
        {
            // lookup candidate in the widget registry to see if a widget of that ID already exists
            // don't keep a new widget registry, use the one that already exists

            // if the id is already in use
            if (<id_is_in_use>)
            {
                continue;
            }

            // don't return directly from here
        }

        if (nextId < 0)
        {
            wrapped = true;
            nextId = WIDGET_ID_RANGE_BEGIN;
        }

        // at this point we know that candidate is OK so we exit our loop
        break;
    }

    return candidate;
}
```

The common case is very fast and only in the (very unlikely) case where we wrap (2 billion widget IDs must be allocated first), do we need to check the widget registry.

I am interested in the results of your PAUSE version. But we already have enough knowledge to know that there are more than 64K of widgets that can be created in a normal environment.

We know this:

1. Unless the -usewidgetid feature is enabled in the 4GL, the widget IDs are not visible directly to 4GL code. We don't have a current need to implement -usewidgetid mode right now. Since the widget IDs are not visible, there are no special requirements of what numbers are assigned.
2. The size of the "address space" of widget IDs is unclear right now, but it is large enough that we don't have to limit it to a 16-bit integer.

More findings for extended stack use:

The loop stopped at 9974 widget creation without delete.

But interesting point is: with widget deletion the process can reach even 32-bit integer limit without any issue and go further(> 6000000 numbers and I have stopped). So I guess the widget ID limitation works only for -usewidgetid option. Without this option there is no such limitation (and even there is no widget ID concept in use). And the only limitation for no -usewidgetid option is the memory/handle resources limit available for application to use.

OK with your approach for implementing.

Only question to be clear. The ID will be incrementing for new widget? Meaning allocation direction from 1 to UPPER\_LIMIT, correct? I'm asking because if we will need -usewidgetid to be implemented further - we will have to rework this code once again (more or less).

The ID will be incrementing for new widget?

Yes.

Meaning allocation direction from 1 to UPPER\_LIMIT, correct?

No. At this time I don't want to set an upper limit. The int will naturally "wrap" to a negative value at the approx 2 billion mark. The algorithm as I specified relies upon that fact.

I'm asking because if we will need -usewidgetid to be implemented further - we will have to rework this code once again (more or less).

I understand.

**#82 - 08/10/2014 06:50 PM - Eugenie Lyzenko**

- File `evl_upd20140810a.zip` added

The update is the new widget ID allocation approach for you to review. Does not include the dynamic widget ID support code developed at this time to separate these two subtask and get working widget ID code first.

**#83 - 08/11/2014 11:08 AM - Eugenie Lyzenko**

The question/clarification for new ID approach update. I've found the class `EventDefinition` is used on both on the server and client sides. The change regarding new widget ID is inside the `lookup()` method. I made assumption this method is calling on the client side only. But may be it will be better to have opportunity to determine if the context is server or client at run time. So we need to implement the call to do this if it is not already implemented. What do you think?

**#84 - 08/11/2014 12:04 PM - Eugenie Lyzenko**

- File `evl_upd20140811a.zip` added

Forgot to add `ThinClient.java` to previous update. Also the javadoc comments added in `WidgetId` class.

**#85 - 08/11/2014 03:43 PM - Eugenie Lyzenko**

- File `evl_upd20140811b.zip` added

This update adds client/server session type checking into `EventDefinition.java`.

**#86 - 08/12/2014 10:56 AM - Greg Shah**

Code Review 0811b

1. In `WidgetId`, the class javadoc states "Every JVM session has it's own set of widget ID.". The use of "JVM session" suggests you are talking about static data that is stored per-JVM instance. Instead, please state this: "The widget ID address space is stored and managed on a context local basis. This means that each user's P2J session maintains an independent set of widget ID allocations."

2. Please move the logic from `WidgetId.SessionArea.allocateId()` into `WidgetId.allocateId()`. `WidgetId.SessionArea.allocateId()` should not exist. The `SessionArea` should only have the `nextId` and `wrapped` members.

3. This code in `allocateId()` doesn't do exactly what is suggested:

```
// protection from infinite loop when there is no free ID to allocate
// we are here on the second wrapping
if (nextId < 0 && wrapped)
{
    throw new RuntimeException("No free ID to allocate!");
}
```

All we know here is that we are wrapping for the second (or a subsequent) time. We do NOT know that we have wrapped twice in a row before returning from `allocateId()`. In other words, we don't know if there are unallocated IDs, so this is not safe. Additional state will be needed to provide this feature. I think it is a good idea, but we just need to code it properly.

4. `WidgetId.possibleValidFrameId()` does not do anything useful in this new approach. Get rid of it.

5. The `WidgetId.allocateWindowID()` can't work for allocating anything other than the `DEFAULT-WINDOW`, which is not sufficient. It breaks code we already have working in GUI. In addition, we aren't keeping any special IDs for window widgets, except for the very first/implicit window created.

Please remove `WidgetId.allocateWindowID()` and just make sure that the default window that is created at startup uses `WidgetId.DEFAULT_WINDOW_ID` (which should be a public member).

6. In regard to the changes for `EventDefinition`, I really don't want this code to have client and server specific implementation code:

```
// determine if the calling environment is the client side or server side
if (ThinClient.getInstance() != null)
{ // got client session we are on the client side
  fId = ThinClient.frameIdFromWidgetId(widgetId);
}
else
{ // not a client considered as server side
  fId = LogicalTerminal.frameIdFromWidgetId(widgetId);
}
```

As a general approach, I do NOT want to add code like this throughout the project. Perhaps we need an approach like the following:

- For features that need to be implemented on both the server and client, we create an API (Java interface).
- We create a factory class that can be used to obtain a helper object that implements that Java interface.
- The startup code for the client (`ThinClient`) and the server (`LogicalTerminal`) would register client-specific or server-specific instances of that helper, which can be used to actually do the lookups needed.

7. In `GenericFrame.getWidgetForId()`, the approach is slow. The `LogicalTerminal.lookupFrameIdFromWidgetId()` is even slower. Please investigate keeping a single server-side registry of all widgets, including windows, frames and all the contained widgets. This should be a map of id to widget. Then we can obtain a widget reference very quickly by lookup into this map. Obtaining a frame ID from a widget ID would involve lookup of the widget and then getting the containing frame out and asking the frame for its ID. This will also allow `LogicalTerminal.isWidgetInUse()` to be done very simply (if the ID exists in the widget registry, then it is in use). The key to making this work is to ensure that we properly add/remove from this registry at all the right times.

**#87 - 08/12/2014 07:08 PM - Eugenie Lyzenko**

5. The `WidgetId.allocateWindowID()` can't work for allocating anything other than the `DEFAULT-WINDOW`, which is not sufficient. It breaks code we already have working in GUI. In addition, we aren't keeping any special IDs for window widgets, except for the very first/implicit window created. Please remove `WidgetId.allocateWindowID()` and just make sure that the default window that is created at startup uses `WidgetId.DEFAULT_WINDOW_ID` (which should be a public member).

I'm looking for the best way to do this. Found the default window is not pushing from server to client. We just create similar objects on both sides with

default ID 999. So I think the solution can be just replacement WindowConfig.SESSION\_WINDOW\_ID with WidgetId.DEFAULT\_WINDOW\_ID in all places that refer to WindowConfig.SESSION\_WINDOW\_ID. And get rid of the WindowConfig.SESSION\_WINDOW\_ID. If the window is dynamic - the ID will be changed on the window registration step as any other widget. Is it OK?

Or if such modification is too deep, we can change the default number for SESSION\_WINDOW\_ID constant in WindowConfig class this way:

```
...
/** Permanent ID of the session window. */
public static final int SESSION_WINDOW_ID = WidgetId.DEFAULT_WINDOW_ID;
...
This allows to minimize the required changes.
```

**#88 - 08/13/2014 06:28 AM - Greg Shah**

I think the solution can be just replacement WindowConfig.SESSION\_WINDOW\_ID with WidgetId.DEFAULT\_WINDOW\_ID in all places that refer to WindowConfig.SESSION\_WINDOW\_ID. And get rid of the WindowConfig.SESSION\_WINDOW\_ID. If the window is dynamic - the ID will be changed on the window registration step as any other widget. Is it OK?

Yes, this sounds like a reasonable plan.

**#89 - 08/13/2014 04:05 PM - Eugenie Lyzenko**

- File *evl\_upd20140813a.zip* added

Working on the point [#7](#) from your note. But have the update to upload meantime.

In this update for review: `FrameldFromWidgetId()` methods has been removed from both `ThinClient` and `LogicalTerminal` because we do not need anymore static interfaces to private methods. Instead the methods `lookupFrameldFromWidgetId()` are made public. The new API interface introduced to handle the ID related requests related to both client and server code - `WidgetIdWorker`. The worker class `WidgetIdHelper` takes the client or server implementation via `initWorker()` static call and stores the reference in session context area (not sure maybe we can just use simple static member for real worker class instead). The both `LogicalTerminal` and `ThinClient` classes implements the single method of the `WidgetIdWorker` interface - `lookupFrameldFromWidgetId()`.

**#90 - 08/13/2014 05:39 PM - Eugenie Lyzenko**

The point to clarify:

Please investigate keeping a single server-side registry of all widgets, including windows, frames and all the contained widgets. This should be a map of id to widget.

In the `LogicalTerminal` there are already several maps that serve similar functions: `frameRegistry`, `sharedFrameRegistry`, `windowRegistry`. Must the

new widget map be a replacement for all of them? Or new map will work in addition to existed ones and old maps and new will exist simultaneously?

**#91 - 08/14/2014 08:37 AM - Greg Shah**

Code Review 0813a

This version is very good.

My only concern is how this will work in certain scenarios like CTRL-C or "single" mode (where the server and client are actually running in the same JVM). To eliminate these things from being a problem I suggest the following changes:

- Create two special classes: ClientIdHelper and ServerIdHelper.
- In each of these classes, have them call a static member (yes, it will have to be put back) in ThinClient or LogicalTerminal to get the ID.
- At startup, an one of these helpers is registered instead of directly storing an instance of ThinClient or LogicalTerminal.

Your approach is cleaner in code but since it has the potential for problems because of some of our use-cases, we need to make it slightly more complex in order to use the static methods.

Other answers:

Must the new widget map be a replacement for all of them?

No.

Or new map will work in addition to existed ones and old maps and new will exist simultaneously?

Yes, it should be in addition and must exist simultaneously.

**#92 - 08/14/2014 11:44 AM - Eugenie Lyzenko**

- File *evl\_upd20140814a.zip* added

This update for review reflects the recent notes to using ThinClient or LogicalTerminal to work with ID. I need to know whether this implementation is acceptable or not.

The widget registry is still under construction.

**#93 - 08/14/2014 02:01 PM - Eugenie Lyzenko**

The question. May be it is better to move ClientIdHelper class to com.goldencode.p2j.ui.chui package(the same place as ThinClient class) and remove import statement rather than having this class in com.goldencode.p2j.ui? What do you think?

**#94 - 08/14/2014 02:05 PM - Greg Shah**

Code Review 0814a

Yes, the approach is correct. Some minor feedback:

1. The ClientIdHelper should be in the com.goldencode.p2j.ui.client package.
2. The default constructor does not need to exist for either the ClientIdHelper or the ServerIdHelper. Java will create an implicit one for us.

The widget registry is still under construction.

Understood.

**#95 - 08/14/2014 02:06 PM - Greg Shah**

May be it is better to move ClientIdHelper class to com.goldencode.p2j.ui.chui package(the same place as ThinClient class) and remove import statement rather than having this class in com.goldencode.p2j.ui?

We are moving all code out of the com.goldencode.p2j.ui.chui package, so I don't want to add new code there now.

**#96 - 08/14/2014 05:12 PM - Eugenie Lyzenko**

- File evl\_upd20140814b.zip added

The next update for your review contains widget ID registry implementation in addition to other notes resolution. The location of the registry is LogicalTerminal. This map can contain every entity that has the allocates ID:

1. Widget or pseudo widget(header) that is adding to the frame in GenericFrame class. So we do not need special method for header widget(getAnyWidgetForIdInt method is now completely the same as getWidgetForId).
2. Frame or shared frame that passes the registration process.
3. Window except default session window.

For now the entity can be removed from this map if frame is being de-registering. If frame is moved from shared to usual state nothing is changed in this map.

The question: the map has data pair <Integer,GenericWidget>. On the other hand we can(and other maps uses this) use both int and Integer as key object. The Integer is a wrap for int. And the question: is it equal to use of the 10 and Integer(10)? Will we get the same value for map.get(10) and map.get(new Integer(10))? Do we need to use only Integer wrap class to work with map?

The update does not contain dynamic widget implementation code.

## #97 - 08/14/2014 06:20 PM - Greg Shah

Code Review 0814b

1. The locking for the widget registry (and frame registries...) needs to be improved. Right now, you can edit the same map simultaneously because different lock objects are being used:

```
synchronized (lt.frameRegistry) in LT.registerFrame()
synchronized (lt.frameRegistry) in LT.replaceFrame()
synchronized (lt.frameRegistry) in LT.releaseSharedFrame()
synchronized (frameRegistry) in LT.deregisterFrameInt()
synchronized (lt.frameRegistry) in LT.registerWindow()
synchronized (lt.frameRegistry) in LT.deregisterWindow()
synchronized (lt.widgetRegistry) in LT.registerWidgetIdObject()
no read protection in LT.isWidgetIdInUse()
no read protection in LT.lookupFrameIdFromWidgetId()
no read protection in LT.getWidgetForId()
no read protection in LT.getAnyWidgetForIdInt()
```

Please add a new data member to LT:

```
private Object registryLock = new Object();
```

Then in all of these places, replace synchronization on frameRegistry or widgetRegistry with registryLock. Add synchronization in the read locations that are not protected today.

2. Memory leak for regular widgets because there is no deregisterWidgetIdObject(). If there is no natural place to put that logic in GenericFrame, then we need to add some extra processing whenever a frame is removed from the registry, to walk all its non-dynamic widgets (header and non-header) and remove them all.

3. I agree we no longer need LT.getAnyWidgetForIdInt(). Just reuse LT.getWidgetForId().

4. In LT.registerWidgetIdObject() there is this code:

```
// simple check for validity
if (widget == null || widgetId <= WidgetId.DEFAULT_WINDOW_ID)
{
    return;
}
```

Put logging here before the return so that we can know if this ever occurs (it shouldn't, but if it does we need to know).

The question: the map has data pair <Integer,GenericWidget>. On the other hand we can (and other maps uses this) use both int and Integer as key object. The Integer is a wrap for int. And the question: is it equal to use of the 10 and Integer(10)? Will we get the same value for map.get(10) and map.get(new Integer(10))? Do we need to use only Integer wrap class to work with map?

It is safe as long as you are only allowing autoboxing using int and Integer. If you ever mix in Long or long, it will not work right. But your code should be safe.

The update does not contain dynamic widget implementation code.

Please do that next work in a separate update. Let's get the current update finished, tested and checked in on its own.

#99 - 08/15/2014 03:15 PM - Eugenie Lyzenko

- File evl\_upd20140815a.zip added

This update for your review includes:

1. Operations with widget registry are now have protection locking
2. Adding code to remove widget ID from registry on frame deregistration stage. GenericFrame does not have obvious places to do this so it is in LogicalTerminal. GenericFrame is used to get the array of the widget ID to delete.
3. Also the special method to deregister single widget has been added to LogicalTerminal. Not it is not used but we will need it on the dynamic widgets handling will be included in code.
4. The common code for LT.getAnyWidgetForIdInt() and LT.getWidgetForId() is moved to LT.getAnyWidgetForIdInt() body because it is private, non-static and used by some methods in LT.
5. Warning log generation has been added to widget register/deregister code.

Several notes:

I have faced with the classes redefinition issues during log implementation. In LT we have included both com.goldencode.util and com.goldencode.p2j.util packages and both have LogHelper classes inside. Is this expected? So to compile LT I had to define LogHelper this way:

```
...
private static final Logger LOG =
    com.goldencode.p2j.util.LogHelper.getLogger(LogicalTerminal.class.getName());
...
```

And this is something that breaks the code writing rules accepted in GCD. May be we need to consider classes renaming to avoid duplications?

Also I had to declare single class import for Logger:

```
...
import java.util.logging.Logger;
...
```

If not to do this the ErrorManager class from java.util.logging conflicts with our same class from com.goldencode.p2j.util. Another point to think about class renaming?

**#100 - 08/16/2014 06:24 PM - Eugenie Lyzenko**

- File evl\_upd20140816a.zip added

This update adds more cleanup for frame deregistration. Looks like we need to remove widgets even when the frame is moving from shared state to private and back. The common code has moved to the separate method. And because the method deregisterFrameContent() is always calling from locked code - it has no lock inside.

**#101 - 08/18/2014 07:49 AM - Eugenie Lyzenko**

Checking the code with frame related testcases I have found potential issues with widget registry. This is related to the shared frame's life cycle. On the one hand we have GenericFrame.createSharedFrame() reflecting to def new shared frame ... Progress statement. And on the other hand GenericFrame.importSharedFrame() reflecting to def shared frame .... The second statement belongs to the external procedure being called from code where first statement is defined. From the LogicalTerminal point of view we just move the frame from sharedFrameRegistry to frameRegistry. The result - we call initialization procedure twice and allocate ID for widget inside the frame also twice(except the frame ID itself).

The example is frame\_driver.p and frame\_user.p test pair.

So the question is it OK to have two copies of widget with different ID, one for frame defined in caller and other - for frame declared in calling procedure? Or we need to clean caller's widget registry entries at the moment when control is passing from caller to calling and clean calling frame widgets when the control is returning from calling procedure to caller?

Another possible approach can be checking inside the widget registry when we allocate new ID. If the given widget is already in registry - we could use already allocated ID instead getting new one.

What do you think?

**#102 - 08/18/2014 09:51 AM - Eugenie Lyzenko**

- File evl\_upd20140818a.zip added

Adding some improvements in this update:

1. Deregister frame when it is destroyed
2. Adding static method to deregister all widgets inside the frame

**#103 - 08/18/2014 01:10 PM - Greg Shah**

Code Review 0818a

1. Please rename the following:

registerWidgetIdObject() to registerWidget()  
deregisterWidgetIdObject() to deregisterWidget()  
getAnyWidgetForIdInt() to getWidgetForIdInt()

2. Please remove getAnyWidgetForId(). It is just a less efficient way to call getWidgetForId(), so we don't need it.

3. The call to LogicalTerminal.deregisterFrame() from inside GenericFrame.destroy() should not be there. The frame actually still exists at that point

and it is only after the client hides the frame that it can be fully cleaned up. See `LT.applyChanges()`. There may be code that assumes the frame is still in the registry. At least on the client side, the IDs are still in use and we cannot allocate these IDs again before the frame is "really gone".

4. The `LogHelper` and `ErrorManager` name collisions are expected.

5. The `deregisterFrameContent()` code should still have the locking implemented. Adding it will not deadlock the code because the lock is already held by that thread. The key reason it makes sense to put it in is to make sure that code added later to call it from somewhere else might not be already owning the lock. Someone that didn't read the code carefully might now realize it. This also means that `deregisterFrameContent()` does not need to hold the lock because it doesn't directly edit or read the protected resources.

6. Shortly, I will write another note about the shared frame processing. I think the current approach you are taking is not correct.

**#104 - 08/18/2014 04:16 PM - Eugenie Lyzenko**

5. The `deregisterFrameContent()` code should still have the locking implemented...

You mean private `deregisterFrameContentInt()` must have lock inside and `deregisterFrameContent` should not have lock? Or both of them must have locks inside?

**#105 - 08/18/2014 04:35 PM - Greg Shah**

In regard to the shared frame processing, it is quite complex. I'm not sure that our basic approach is 100% correct. That means your changes will have to be crafted carefully and tested well.

The basic idea of shared frames is that a 4GL program can create a frame that can be used by multiple programs. Those programs are "further down" the call-stack. They "import" the shared frame by its text based name that was used in the source code. The strange thing in the 4GL is that the same basic frame must be defined in all programs that are going to use it. So you can't define it in only 1 place. If 2 programs are going to share a frame, as long as both programs reference the same frame name AND both those frame definitions are structurally the same (the same list of widgets), then the frames can be shared. If I recall correctly, the two frame definitions do NOT have to be identical. Some configuration can be different. I don't know if we have fully explored the possibilities there.

When we implemented this, we found that when a called program uses a shared frame, it will change that frame's state. But some of the state does NOT change! We have to carefully preserve and copy back the state that changes but not then state that doesn't change.

I will refer to the shared frame registry as `sharedframereg` and the normal frame registry as `framereg`. The following is describing how the current P2J does things. It is not describing your changes.

1. When a non-shared frame is created, the `GF.createFrame()` is called. This creates the proxy object `GF.createFrameProxy()` which uses the Java dynamic proxy support to create a runtime instance that implements the list of given Java interfaces. Then it calls `GF.initializeNewFrame()` which really just calls `GF.initialize()`. `GF.initialize()` calls `LT.registerFrame()` which allocates the frame ID and saves the ID in the `framereg`. Then `GF.initialize()` calls `GF.coreInitialize()` which creates the widget lists, the field group and allocates IDs for all the widgets. It also "cross-links" the widgets back to their containing frame. `GF.initialize()` then creates a new screen buffer. The last thing that `GF.createFrame()` does is call the `GF.setupFrame()` which just uses reflection to call the frame interface's `setup()` method that forces the proper configuration into the frame and its widgets.
2. Before that frame can be used, the converted business logic will call `openScope()` to associate the frame with the block processing of the program. Some frame behaviors are dependent upon the scoping notifications that are provided by the blocks to which they are scoped.
3. When a new shared frame is created, we call `GF.createSharedFrame()`. This goes through the normal processes for `GF.createFrame()` (it allocates IDs for the frame and the widgets, registers them...), but it also will save the frame in the `SharedVariableManager` so that it can be "imported" later. If an already existing frame of the same name is found in the `SharedVariableManager`, that frame will be replaced in the `SharedVariableManager` but the previous frame instance will be returned. When this instance is not null, we call `LT.releaseSharedFrame()` to unconditionally clear the old frame from the `sharedframereg`. At this point, the new frame only exists in the `framereg`.

4. `LT.releaseSharedFrame()` is the only way that shared frames actually "go away" permanently. Until that is called, a shared frame will exist in either the `framereg` or the `sharedframereg`. Since `LT.releaseSharedFrame()` is only called from inside `GF.createSharedFrame()`, this means that we only permanently get dispose of a shared frame when we create a new shared frame of the same name. This suggests that we have a memory leak, since there seems to be no particular point in time when we otherwise clear shared frames out of the `sharedframereg`. I don't know the reason that we keep these frames around after they have been otherwise disposed of. Constantin: do you know?
5. The program that created the shared frame can use it like a normal frame (opening a scope and then referencing it in client calls like `display()`). It really is a normal frame.
6. At some point a called program (possibly many levels down the call stack) will reference the shared frame name and call `GF.importSharedFrame()`. This will NOT call the `GF.createFrame()` directly. Instead, it first calls the `GF.createFrameProxy()` to get a new dynamic proxy instance that can be used to reference the instance of the specific frame definition that is known to this called program. That instance can be slightly different from the original frame, but it must be structurally the same. This instance is the "newframe". The previously shared frame ("oldframe") is then looked up in the `SharedVariableManager`. We then read the oldframe's frame ID and call `LT.activateSharedFrame()`. `LT.activateSharedFrame()` only exists to move any existing frame (with that same ID) from the `sharedframereg` to the `framereg`. See below for how a frame can have been moved into the `sharedframereg`. We then save off the visibility state of the old frame. At this point we create an instance of the `SharedFrameCleaner`, passing it references to the old and new frames. Please note that at this point we have NOT allocated any new ID for the new frame.
7. The `SharedFrameCleaner` exists to make a partial copy of the original frame state into the new frame. It also calls `LT.replaceFrame()` to put the new frame into the `framereg` instead of the old frame instance. This `SharedFrameCleaner` instance is also registered with the `TransactionManager` so that when the called external procedure exits, the `SharedFrameCleaner` will have its `finished()` method called. In that method it copies some state from the new frame back into the old frame. It also calls `LT.replaceFrame()` to put the old frame into the `framereg` instead of the new frame instance. It is during the `SharedFrameCleaner` constructor that the new frame's ID is set to be the same as the old frame's ID.
8. After the `SharedFrameCleaner` is instantiated/registered with the `TransactionManager`, the `GF.importSharedFrame()` will call `GF.coreInitialize()`. This will assign new IDs for the widgets, but because of the way we do things, those widgets will have the same IDs in the new frame that the corresponding widgets had in the old frame.
9. At the end of the method `GF.importSharedFrame()` will restore widget visibility from its saved off state and then call `GF.setupFrame()` to finish the configuration of the new frame. At that point, the new "imported" frame instance can be used by the called external procedure.
10. Although scope-related block events can cause `GF.destroy()` to be called, that just calls `LT.destroyFrame()` which for normal frames will just add the frame to a list that the client can mark as eligible for destruction (dialog boxes are handled synchronously). It does NOT and should not remove the frame from the `framereg` at this point.
11. Once the client has actually hidden the frame, it will notify the server and in the `LT.applyChanges()` method, the frame will be removed from the `framereg`. The strange thing is that shared frames will be added back into the `sharedframereg` by this code.

Some thoughts on your implementation:

- I think the `sharedframereg` is NOT needed after your changes. It is only used to make sure that shared frame IDs are not allocated in `LT.findNextID()`. Since your widget registry already takes care of this, we can simplify our code by removing all references to the `sharedframereg` (`LT.sharedFrameRegistry`).
- If we have to keep the current "feature" of leaving the shared frames around indefinitely, then we will just keep them in the widget registry directly. That would mean that `LT.deregisterFrameInt()` would only remove the frame from the widget registry if it is NOT shared.
- We can remove the `LT.activateSharedFrame()` method completely.
- The `LT.releaseSharedFrame()` would be changed to remove the shared frame from the widget registry instead of the `sharedframereg`.
- The processing of `LT.replaceFrame()` probably needs to remap the new frame's contained widgets into the widget registry in place of the old frame's. Otherwise the old frame's widgets can be referenced by accident. I don't think we need to remove the IDs, we must replace them.

Based on the above points, please ask any questions you may have.

**#106 - 08/18/2014 04:37 PM - Greg Shah**

Eugenie Lyzenko wrote:

5. The deregisterFrameContent() code should still have the locking implemented...

You mean private deregisterFrameContentInt() must have lock inside and deregisterFrameContent should not have lock?

Yes. Please move the locking from deregisterFrameContent() into deregisterFrameContentInt.

**#107 - 08/18/2014 05:43 PM - Eugenie Lyzenko**

I think the sharedframereg is NOT needed after your changes. It is only used to make sure that shared frame IDs are not allocated in LT.findNextID(). Since your widget registry already takes care of this, we can simplify our code by removing all references to the sharedframereg (LT.sharedFrameRegistry).

The problem here is sharedframereg is a map of <id, GenericFrame> while widget registry is a map of <id, GenericWidget>. So we can not store full data for GenericFrame object in widget registry, only FrameWidget part of it.

**#108 - 08/18/2014 07:22 PM - Eugenie Lyzenko**

11. Once the client has actually hidden the frame, it will notify the server and in the LT.applyChanges() method, the frame will be removed from the framereg. The strange thing is that shared frames will be added back into the sharedframereg by this code.

Correct me if I'm wrong. The sharedframereg is used to "reserve" ID for possible further usage of this frame as shared in some next code (we can not predict whether we encounter this frame or not in future), correct? And we do not need the GenericFrame instance data of this object map, just remember this ID was used by "some" shared frame before. That's why I think it is moved into sharedframereg on the deregistration step instead of removing - to be able to move it back to framereg and replace with new internal data.

This means if we preserve this approach we will not release this shared frame related resource until LT session ends. This memory leak is a cost of ready to use shared frame pool.

**#109 - 08/18/2014 07:45 PM - Eugenie Lyzenko**

The processing of `LT.replaceFrame()` probably needs to remap the new frame's contained widgets into the widget registry in place of the old frame's. Otherwise the old frame's widgets can be referenced by accident. I don't think we need to remove the IDs, we must replace them.

You mean we need the new widgets to be associated with the old IDs?

**#110 - 08/18/2014 10:41 PM - Greg Shah**

The problem here is `sharedframereg` is a map of `<id, GenericFrame>` while widget registry is a map of `<id, GenericWidget>`. So we can not store full data for `GenericFrame` object in widget registry, only `FrameWidget` part of it.

We can always use a map of `<int, Object>` instead.

The `sharedframereg` is used to "reserve" ID for possible further usage of this frame as shared in some next code (we can not predict whether we encounter this frame or not in future), correct?

I think so.

And we do not need the `GenericFrame` instance data of this object map, just remember this ID was used by "some" shared frame before.

No, `LT.activateSharedFrame()` actually does move the `GenericFrame` instance over to the `framereg`. That instance is the old frame and although it "gets replaced", it also gets restored by the `finished()` method of the `SharedFrameCleaner`.

This means if we preserve this approach we will not release this shared frame related resource until LT session ends. This memory leak is a cost of ready to use shared frame pool.

Yes, I think we do have a memory leak. But this isn't about having a pool of ready-made resources. These resources are very specific frames that may be active. The problem is that some are not active but we keep them around.

Anyway, don't worry about the memory leak for now. We will discuss that further later this week when Constantin gets back. For now, focus on getting the new widget registry implemented properly with the same level of features as we already support.

You mean we need the new widgets to be associated with the old IDs?

Yes, but we need to be able to restore the old widgets back into the registry too (just like the old frame is restored, during the `finished()` method of the `SharedFrameCleaner`).

**#111 - 08/19/2014 05:37 AM - Eugenie Lyzenko**

We can remove the LT.activateSharedFrame() method completely.

In this case we need to move the work this method does (updating framereg with data of the frame that become active) into LT.replaceFrame(), correct?

**#112 - 08/19/2014 05:55 AM - Greg Shah**

Eugenie Lyzenko wrote:

We can remove the LT.activateSharedFrame() method completely.

In this case we need to move the work this method does (updating framereg with data of the frame that become active) into LT.replaceFrame(), correct?

No, I don't think it is needed at all. There is no need for the sharedframereg, so there is no need to move frames between the framereg and sharedframereg. The reason for the sharedframereg is to ensure that:

1. The shared frame ID is not reused while it can still be active.
2. The shared frame is not garbage collected.

The fact that the frames will always live in the widget registry is enough to ensure both of these things. The point to replaceFrame is to temporarily save (swap) and restore (swap back) the old shared frame instance in the framereg with the new frame instance. Please look at the SharedFrameCleaner carefully to see that the constructor does the save off (old to new) and the finished does the restore (new back to old). This has nothing to do with the old sharedframereg functionality.

**#113 - 08/19/2014 03:42 PM - Eugenie Lyzenko**

- File evl\_upd20140819a.zip added

This update for your review includes:

1. Methods renaming for `registerWidgetIdObject()`, `deregisterWidgetIdObject()`, `getAnyWidgetForIdInt()`
2. `getAnyWidgetForId()` removed
3. Lock has been moved to internal private method `deregisterFrameContentInt()`
4. Frame and content deregistration now in `LT.deregisterFrameInt()`
5. The `LT.sharedFrameRegistry` has been completely removed.
6. No more `LT.activateSharedFrame()`
7. All shared frames handling now goes via `widgetRegistry`
8. `widgetRegistry` can take `GenericFrame` as value, the code for `getWidgetForIdInt` has changed to handle different data type and to be able to react to invalid ID as input parameter.
9. `GenericFrame` has new method to restore static widget registry after shared frames "swap"
10. `LT.replaceFrame()` now remove static widgets for "old" frame from registry.

General notes.

- The shared frames importing uses `SharedFrameCleaner` constructor where we replace "old" frame with "new". In the constructor "old" widgets for the given frame are removed from registry. After this the method `GF.importSharedFrame()` calls `coreInitialize()` which in turn take new ID for widgets in "new" frame and fills the widget registry. I think this is OK because from formal point of view we have new frame with new widgets.
- When it is time to get "old" frame back the `SharedFrameCleaner` in `finished()` makes `LT.replaceFrame()` that removes widget that were put for "new" frame and makes one more call `GF.restoreWidgetRegistry()`. This restores "old" widgets(with old ID) in the widget registry

Please comment if this approach is OK or not.

#### #114 - 08/19/2014 05:32 PM - Greg Shah

Code Review 0819a

I like it.

Does it work? :) Please test it with standalone testcases, including shared frames. If that all works, then put it into runtime regression testing.

#### #115 - 08/19/2014 06:46 PM - Eugenie Lyzenko

Does it work? :) Please test it with standalone testcases, including shared frames.

Yes, but there is one little thing I've missed here. Initially I verified code with shared frames test so it works. But there is one artificial frame that exists on the client side that interfere with new ID approach. Consider `ThinClient.INPUT_FRAME_ID`. This serves the `tinyInput` editor for messages that allow to change the value. This editor frame has only `FillIn` widget inside and consumes two sequential ID, `INPUT_FRAME_ID` and `INPUT_FRAME_ID+1`. In our code it 1 and 2. And this immediately conflicts with ID reserved to default window.

I think these widgets do not have the server side counterpart because they exist only at editing time. So what we can do in this situation? To reserve two more ID (2 and 3) for this editor starting `WIDGET_ID_RANGE_BEGIN` from 4? Or to reserve two numbers somewhere in the end(`0x7FFFFFFF-2` and `0x7FFFFFFF-1`). Because it can be too expensive to consume dynamic ID from server side(in case of frequently using this tiny editor we will waste the ID numbers). What do you think?

**#116 - 08/19/2014 06:54 PM - Eugenie Lyzenko**

Also I guess this artificial frame and fill-in will neither be in widget registry not the part of server side frame. So looks like we will never know if it currently exists and ID occupied.

**#117 - 08/20/2014 08:18 AM - Greg Shah**

I think these widgets do not have the server side counterpart because they exist only at editing time.

Yes.

To reserve two more ID (2 and 3) for this editor starting WIDGET\_ID\_RANGE\_BEGIN from 4?

Yes, this seems the easiest choice.

Because it can be too expensive to consume dynamic ID from server side(in case of frequently using this tiny editor we will waste the ID numbers).

As far as I know, we have no such need today.

Also I guess this artificial frame and fill-in will neither be in widget registry not the part of server side frame. So looks like we will never know if it currently exists and ID occupied.

I think that is OK.

**#118 - 08/20/2014 09:42 AM - Eugenie Lyzenko**

- File evl\_upd20140820a.zip added

This update for your review moves the INPUT\_FRAME\_ID constant to WidgetId class. The WIDGET\_ID\_RANGE\_BEGIN now starts from 4. And I have updated the javadoc for WidgetId to reflect implementation change.

If you do not mind I'll start the runtime regression testing process. If you think we need to rename INPUT\_FRAME\_ID to something more client specific - let me know please.

**#119 - 08/20/2014 09:46 AM - Greg Shah**

Code Review 0820a

I like the changes. Well done.

If you do not mind I'll start the runtime regression testing process.

Yes, please do.

If you think we need to rename INPUT\_FRAME\_ID to something more client specific - let me know please.

No, the Javadoc changes suitably describe the implementation.

**#120 - 08/20/2014 10:32 AM - Eugenie Lyzenko**

- File evl\_upd20140820b.zip added

Sorry, I did not remove ClientIdHelper from old location, so 0820a has two files with same class. Repackaged in 0820b. Regression testing is in progress.

**#121 - 08/20/2014 04:18 PM - Eugenie Lyzenko**

The update certainly has a regressions in frame ID handling (on client side I guess). After main syman menu displaying pressing v causes on client side:

```
...
java.lang.RuntimeException: Invalid frame ID 118
    at com.goldencode.p2j.ui.chui.ThinClient.getFrame(ThinClient.java:10928)
    at com.goldencode.p2j.ui.chui.ThinClient.getChanges(ThinClient.java:10642)
    at com.goldencode.p2j.net.Protocol.attachChanges(Protocol.java:274)
    at com.goldencode.p2j.net.Queue.enqueueOutbound(Queue.java:810)
    at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:765)
    at com.goldencode.p2j.net.Conversation.block(Conversation.java:319)
    at com.goldencode.p2j.net.Conversation.waitMessage(Conversation.java:257)
    at com.goldencode.p2j.net.Queue.transactImpl(Queue.java:1128)
    at com.goldencode.p2j.net.Queue.transact(Queue.java:585)
    at com.goldencode.p2j.net.BaseSession.transact(BaseSession.java:179)
    at com.goldencode.p2j.net.HighLevelObject.transact(HighLevelObject.java:163)
```

```

at com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(RemoteObject.java:1423)
at com.goldencode.p2j.net.InvocationStub.invoke(InvocationStub.java:97)
at com.sun.proxy.$Proxy4.standardEntry(Unknown Source)
at com.goldencode.p2j.main.ClientCore.start(ClientCore.java:264)
at com.goldencode.p2j.main.ClientCore.start(ClientCore.java:93)
at com.goldencode.p2j.main.ClientDriver.start(ClientDriver.java:227)
at com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:423)
at com.goldencode.p2j.main.ClientDriver.process(ClientDriver.java:112)
at com.goldencode.p2j.main.ClientDriver.main(ClientDriver.java:290)
...

```

And the client silently exits to command prompt.

The server side has multiple:

```

...
[08/20/2014 12:05:08 EDT] (com.goldencode.p2j.ui.LogicalTerminal:WARNING) Attempt to request widget for invalid ID!
...

```

messages.

Looking for problematic code. Looks like old ID implementation still exists somewhere on the client side.

## #122 - 08/21/2014 08:49 PM - Eugenie Lyzenko

- File *evl\_upd20140821a.zip* added

Looks like I've found the root cause. It really the interference of new ID approach and old one. The method is `WidgetRegistry.removeFrame()`. Consider this:

...

```

int base = frame.config().getID();
Widget comp = frame;

for(int ndx = base; comp != null; ndx++)
{
    comp = widgetList.remove(new Integer(ndx));
}
...

```

This code worked in previous approach because we had frame ID like N000 and 999 widgets for every frame: N001-N999. So while we had up to N998 widgets in the frame the `ndx++` will retrieve non-existed widget and the loop stops at the last widget for given frame. But now when we have sequence of the increasing numbers this code will destroy all widgets in the client's registry from base to the end of the list(including possible other

frames and so on). Even in old approach having 999 widget for the frame was potentially dangerous because the next number could be the valid frame ID.

So the update for your review fixes this problem. Also merged with the recent(10600) code base.

I have tested this fix with simple MAJIC screens and it works. So if you do not have notes, I'll start the regression testing.

Another question I asked myself while debug this issue. When we deregister frame content all widget must be deregistered including possible frames? Or if some frame is a part of another frame we need to consider the situation when we have to keep that frame registered when removing other widgets from parent frame? What do you think? There is nothing to worry about here?

#### **#123 - 08/22/2014 06:42 AM - Constantin Asofiei**

Greg Shah wrote:

1. LT.releaseSharedFrame() is the only way that shared frames actually "go away" permanently. Until that is called, a shared frame will exist in either the framereg or the sharedframereg. Since LT.releaseSharedFrame() is only called from inside GF.createSharedFrame(), this means that we only permanently get dispose of a shared frame when we create a new shared frame of the same name. This suggests that we have a memory leak, since there seems to be no particular point in time when we otherwise clear shared frames out of the sharedframereg. I don't know the reason that we keep these frames around after they have been otherwise disposed of. Constantin: do you know?

No. I think this was overlooked from the initial implementation.

1. The program that created the shared frame can use it like a normal frame (opening a scope and then referencing it in client calls like display()). It really is a normal frame.

The key point here is that this is the same frame (i.e. the same resource, with the same handle ID) as the original frame defined via DEFINE NEW SHARED FRAME. In our current implementation, a new frame resource is created by each DEFINE SHARED FRAME instance, and the handle IDs will be different for the master and the child frame definitions.

#### **#124 - 08/22/2014 08:27 AM - Greg Shah**

Code Review 0821a

I'm fine with the changes. Let's get this through testing. You can fix the memory leak in the next version.

When we deregister frame content all widget must be deregistered including possible frames? Or if some frame is a part of another frame we need to consider the situation when we have to keep that frame registered when removing other widgets from parent frame?

Frames cannot contain other frames, so this is not a problem.

#### #125 - 08/22/2014 04:14 PM - Eugenie Lyzenko

Two more issues has been found during testing:

1. NullPointerException in code LogicalTerminal.deregisterFrameContentInt():

```
...
    int[] wId = frameToProcess.getStaticWidgetIds(true);

    synchronized (registryLock)
    {
        for (int i = 0; i < wId.length; i++)
        {
            ...
        }
    }
}
```

sometimes wId == null because there is nothing to delete. The suggested fix is:

```
...
    int[] wId = frameToProcess.getStaticWidgetIds(true);

    synchronized (registryLock)
    {
        // determine widget numbers to remove and protect from NullPointerException
        int widSize = (wId != null) ? wId.length : 0;

        for (int i = 0; i < widSize; i++)
        {
            ...
        }
    }
}
```

2. The other issue is related to the ScreenBuffer class. We have the state[] to store the widget status. But the old approach makes this calculation to reference byte value, if widget has id, status is: state[id - frameId - 1]. Now this is not correct because we use simple sequential ID approach(this can work in some very simple cases but for sure the current widget ID is not related to frame ID). In testing cycle this causes array out of range exception in ScreenBuffer.mergeChanged() code.

So I would suggest to use another map <Integer, Byte> as status collection to store status byte for every widget ID. And get rid of all inline ID calculation in code.

**#126 - 08/25/2014 10:58 AM - Greg Shah**

The suggested fix is:

I don't see any reason to enter the synchronized block. Entering it obtains the lock for no reason, which is costly. Just put this code above the synchronized block:

```
if (wId == null)
{
    // nothing to do
    return;
}
```

So I would suggest to use another map <Integer, Byte> as status collection to store status byte for every widget ID. And get rid of all inline ID calculation in code.

OK.

**#127 - 08/27/2014 05:40 PM - Eugenie Lyzenko**

- File *evl\_upd20140827a.zip* added

This update for review includes:

1. Several fixes for `ThinClient.(get|set)ScreenBuffer()` and `GenericFrame.getScreenBufferFromCache()` methods to clean up the manual ID calculation.
2. Big rework for `ScreenBuffer` class. Now it is based on map to store the status byte.
3. Merge with the recent code base.

I have several notes/questions. Will publish later today.

**#128 - 08/27/2014 06:51 PM - Eugenie Lyzenko**

Some considerations and questions to discuss for inline widget ID calculation in current code. I have found new places not yet handled in update.

1. Consider the code `src/aero/timco/majic/security/LoginClient.java`. It has ID assignments completely different from new approach (and moreover it differs from old one too). Currently the update 0827a stops the client execution with `NullPointerException` in `ThinClient.setScreenBuffer()` method:

```
...
    // get the widget
    Widget comp = registry().getComponent(id);

    if (frame.isDown())
        comp = frame.getField(comp);

    config = (ComponentConfig) comp.config(); << Here we have null pointer
...
```

The login screen has frames with ID 100 and 200 and widgets inside - 1xx and 2xx. These numbers are hardcoded on the client side (like a tiny input message editor).

The question. Is it possible to debug this login screen on local system with `p2j/testcases/simple/server` environment? I tried to do remote debugging attaching to 2190 client on devbox01 but I can not connect too early due to huge delays in this debug mode to find out why we have null widgets in registry? I guess something is wrong in new `ScreenBuffer` class but to isolate the issue I need to debug MAJIC login screen.

2. The class `ScreenDefinition` from `p2j/ui/*` has method `addConfig()` -> `addConfigInt()` that adds component config for new widget into the current screen definition. And we have inline ID calculation here too:

```
...
    if (wid == -1)
    {
        // auto-generate an ID
        while (widgets.get(new Integer(++last)) != null) { };

        config.setID(last);
        wid = last;
    }
...
```

looks like this code is for pseudo widgets like skip or space. Anyway it can produce conflicts with new ID approach when one ID will be assigned to two or more different widgets.

If we can be sure this code is executing only on the server side - we can just use `WidgetId.allocateId()` instead searching for the last widget in the map and use next ID. But what if the code is executing on the client side too (as for initial MAJIC login screen)?

What I would suggest.

- As we discussed all ID management should be done in one single place. Only on the server side. There can be exception for short lived tiny editor widget (not sure about initial login screen). The client side must only consume ID taken from server.

- In the case the client needs new ID for widget that still not having ID - the client must request server to allocate new ID. This ID consumes on the server side and passes back to the client.

What do you think? I thought we can avoid such complicated schema with client-server calls but looks like getting rid of the inline ID calculation on client side does not let us the choice.

#### #129 - 08/27/2014 10:05 PM - Eugenie Lyzenko

Found a way to debug login screen locally by tunneling server port 3434 and using -i1 instance to remote connection to my -i100 server instance.

#### #130 - 08/28/2014 11:03 AM - Greg Shah

Code Review 0827a

1. In ScreenBuffer, you chose to start with an empty state map instead of one populated by UNINITIALIZED values. This means that in many places, we now have code like this:

```
byte stateCurr = (status != null) ? status.byteValue() : UNINITIALIZED;
```

This code is everywhere that we read out of the map, because of the "not in the map means UNINITIALIZED" approach. This clutters up the code. More concerning is that it makes it easy to add errors later if someone adds code to read from the map without this protection.

There is a benefit to your approach, which is it takes less memory when widgets are UNINITIALIZED. To keep this benefit, but make it safe please do the following:

- create helper methods (like resetState()) to handle all direct access to the state map
- add text to the javadoc for the state map that it must not be accessed directly, but rather only via the helper methods

I think this will also have the benefit of making the state map usage cleaner since there will be more code that is shared. Move as much of the code as possible into the shared helpers so that the callers are as simple as possible.

2. The ScreenBuffer.resetState() name suggests that the entire state byte for the given ID will be cleared, but it doesn't reset all state. It bitwise ANDs the state for a given ID with the given mask and saves it back into the map. The caller normally passes a mask that has all flags set except the one(s) to reset. I like the fact that you created a helper method to do this. I just want it named something else. Perhaps it is best to be descriptive like bitwiseAndState().

3. Why do you have buffer.size() < size as a condition in ScreenBuffer.isValidId()?

#### #131 - 08/28/2014 11:19 AM - Greg Shah

It has ID assignments completely different from new approach(and moreover it differs from old one too).

LoginClient is a temporary and hard-coded "user" of our ThinClient environment. It is not driven by the server side. In fact, there is no server side of the UI for LoginClient. In other words, there is no LogicalTerminal yet. It won't normally be running BUT it does need to be compatible.

It seems to me that it can allocate its own widget IDs (and register them as needed) so long as it properly cleans everything up.

- As we discussed all ID management should be done in one single place. Only on the server side. There can be exception for short lived tiny editor widget(not sure about initial login screen). The client side must only consume ID taken from server.

I agree with this. If we can "fake out" the case for the initial login screen, then this should work fine.

- In the case the client needs new ID for widget that still not having ID - the client must request server to allocate new ID. This ID consumes on the server side and passes back to the client.

What do you think? I thought we can avoid such complicated schema with client-server calls but looks like getting rid of the inline ID calculation on client side does not let us the choice.

Does the `ScreenDefinition.addConfig()` ever get called on the client when `wid == -1`? If not, then we don't need any complicated callbacks to the server.

For the login screen, we can't call the server back anyway because it doesn't exist. We just need to add whatever state is needed to `ThinClient` and other client locations to make things work. Then as part of login screen cleanup we need to remove that state.

**#132 - 08/28/2014 11:48 AM - Eugenie Lyzenko**

3. Why do you have `buffer.size() < size` as a condition in `ScreenBuffer.isValidId()`?

Yes I had to put the describing comment in the source file. The idea is:

- When we create screen buffer we know how exactly much widgets will be in the class, the size variable covers this.
- But we have no widget values initially, the maps(including state) are empty.
- Also we do not know what will be widgets ID stored in this map. And the ID can be arbitrary, no predefined order expected.

So to verify incoming ID we can check if the ID is already in the buffer. This means the ID is correct. But it is possible for valid ID to not be in screen buffer at given time. This means the screen buffer is waiting to put new widget and has the "slot" in map to store the new number.

If all "slots" in buffer map are occupied(`buffer.size() == size`) and the new incoming ID is not in this filled map - I guess we can consider the new ID to handle is "invalid", looks like we are trying to handle the widget not belonging to this frame. If we replace old widget with new one(with new ID) for given frame - we need to remove old widget data first. Then we can put the widget with new ID to the free "slot".

That's idea.

**#133 - 08/28/2014 02:29 PM - Constantin Asofiei**

Eugenie Lyzenko wrote:

1. Consider the code `src/aero/timco/majic/security/LoginClient.java`.

Please check all Authenticator implementations for how the widget IDs are computed.

**#134 - 08/28/2014 05:18 PM - Eugenie Lyzenko**

Please check all Authenticator implementations for how the widget IDs are computed.

Except `GenericFrame`(server side) and `ScreenDefinition` itself the calls to screen definition:

`LoginClient`

uses two frames:

1. One with ID 100, of size 18 to display timco logo
2. Another with ID 200, of size 3 to handle `user(201)/password(203)` pair. Here the skip between widgets and one info string: "Report any problems to the GSO IT Help Desk at extension: 3062." using ID 205. And this is very strange because the size is 3 and old `ScreenBuffer` implementation does not allow to put this ID into buffer. But we see this string on majic login. This is strange.

`p2j/security/DatabaseAuthenticationHook.java`

frame ID is 100, of size 11

`p2j/security/DefaultLoginPanel.java`

frame ID is 100, size is 3 widgets

All these cases use `ScreenDefinition.addConfig()` to place the component, skips require the ID to assign (have -1 initially).

So if these client side calls are in time there is no `LogicalTerminal` and completely isolated from server code and further client code - all we need is to properly clean up after finish, correct?

We need to hardcode all ID, for these cases, including for skip/space pseudo widgets. To not pass the widget with -1 ID to `ScreenDefinition.addConfig()` avoiding ID interference.

Maybe it will be good to reserve some ID range for such purposes, say first 499 or smaller to completely separate usual ID allocation logic from special hardcoded ID usage, starting dynamic ID allocation outside this special range?

**#135 - 08/28/2014 08:05 PM - Eugenie Lyzenko**

- File `evl_upd20140828a.zip` added

Update for review includes:

1. Renaming `ScreenBuffer` helper method to bitwise AND the state.
2. Adding helper methods to work with the state map. The public state iterator getter has been removed to exclude any possibility to directly work with map.
3. `ScreenDefinition` class added with change that removes inline widget ID calculation, getting ID from `WidgetId` class.

Continue working. Need to fix `LoginClient` to be able more debugging.

**#136 - 08/29/2014 09:09 AM - Greg Shah**

all we need is to properly clean up after finish, correct?

You may also need these widgets to be in the `WidgetRegistry` before starting to display or edit the frames. I'm not sure what other "setup" code will be needed, but there will be a "mirror" set of cleanup for each setup item.

We need to hardcode all ID, for these cases, including for skip/space pseudo widgets. To not pass the widget with -1 ID to `ScreenDefinition.addConfig()` avoiding ID interference.

Yes.

May be it will be good to reserve some ID range for such purposes, say first 499 or smaller to completely separate usual ID allocation logic from special hardcoded ID usage, starting dynamic ID allocation outside this special range?

No, let's avoid this for now. As long as you setup and cleanup properly, we don't ever need to know about these because they are gone by the time the "full client" starts.

**#137 - 08/29/2014 09:17 AM - Greg Shah**

I am fine with the changes.

#138 - 08/29/2014 05:54 PM - Eugenie Lyzenko

- File evl\_upd20140829b.zip added
- File evl\_upd20140829a.zip added

These two updates for review contain fixed login related classes, including one from timco majic related sources. I made debug for login panel and can confirm after login frame dismissed the WidgetRegistry used by ThinClient properly cleans with current LoginClient.clientFinalize(), the widget list is empty before main application starts after login.

#139 - 08/29/2014 06:09 PM - Eugenie Lyzenko

Unfortunately at least one regression I have already found. Consider pressing 'V' from main menu.  
The current screen is:

```
...
08/29/2014          SYMAN SERVICE ORDERING          17:56:26
+----- Repair Station: TIPR833K  TIMCO - GREENSBORO      ----+
|S/O: 101633  Gen. S/O: 101633  Customer: 101          Stat: I (Invoiced)
|EstArrv: 09/07/2001  WrkDate:          S-Shft: 0    Site: 1  GSO
|EstDprt: 10/10/2001  WrkTime: 00:00    Cur-Shft: 0    Apprv: Auto N Web N Agg Y
...
```

Expected screen is:

```
...
07/30/2014          SYMAN SERVICE ORDERING          15:20:32
+----- Repair Station: TIPR833K  TIMCO - GREENSBORO      ----+
|S/O: 101633  Gen. S/O: 101633  Customer: 101          Stat: I (Invoiced)
|EstArrv: 09/07/2001  WrkDate:          S-Shft:       Site: 1  GSO
|EstDprt: 10/10/2001  WrkTime: 00:00    Cur-Shft: 0    Apprv: Auto N Web N Agg Y
...
```

The field value for S-Shft: is expected to be initially hidden while we have 0 shown. I suggest this is a kind of state mismatch with new reworked ScreenBuffer implementation for this widget(should be uninitialized but has changed or something like this). Debugging issue. If someone has the info where to dig this please let me know.

**#140 - 09/01/2014 04:03 AM - Eugenie Lyzenko**

- File *evl\_upd20140901a.zip* added

The next update for your review.

Looks like I've found the issue resolution. We need to initialize the state map for every widget in a frame, not depending whether we have the widget value itself. All we need to know is the all ID set for widgets in a frame. This is true after the frame has been completely initialized and all widget has the ID allocated. The method to get ID set iterator has been restored for ScreenBuffer class. In ThinClient now we iterate through the set map to make the screen buffer to set. So we are taking into account the uninitialized widget even if the value is not yet in the screen buffer.

Anyway the regression with SYMAN SERVICE ORDERING is gone.

Also the GenericFrame.remainingIds() has been fixed to get rid of the inline ID calculation. Another minor fix is to take into account the widget map in LogicalTerminal can have both GenericFrame and GenericWidget objects. This is important when getting frame ID from widget ID.

**#141 - 09/01/2014 02:21 PM - Greg Shah**

Code Review 0829a/b and 0901a

I am fine with the changes.

**#142 - 09/01/2014 05:04 PM - Eugenie Lyzenko**

- File *evl\_upd20140901b.zip* added

Another update today for review. I've faced another issues with lost UNINITIALIZED state. In this update all constructors in GenericFrame and ThinClient(except tiny input editor case) to use new one(taking ID as initial array). The main current issue is the difference from old approach when the state already exists even when there is no widget value.

So we need to use the state map as master map to inspect the ScreenBuffer. Because ThinClient should check state of all widgets in a given frame, not only ones need to display. Also I have changed the map to enumerate ID for all state resetting methods in ScreenBuffer. To base on buffer value is not enough to properly reset all state entries in buffer. Also the method ScreenBuffer.reset() should keep the state map but set the state value to UNINITIALIZED.

Another word approach I've started from was inconsistent. We need to have the state map value for every widget in every time the frame exists. To implement this in new approach we need to know the all widget IDs at the time the new ScreenBuffer is creating. Potentially this can create the problems with further dynamic widgets implementation but for now we need the current approach to be consistent and get regression test passed.

So I'm going to run runtime regression tests to check if all issues are now solved.

**#143 - 09/03/2014 09:05 AM - Greg Shah**

Code Review 0901b

The changes look good and they do look more correct.

My only question is whether the `ScreenBuffer(int frameId, int numWidgets)` constructor is ever safe to use in the new approach? If not, shouldn't we get rid of it?

**#144 - 09/03/2014 07:22 PM - Eugenie Lyzenko**

Status update.

The next regression I've faced is in PO adding screen. This is the serious issue with shared frames behavior with new ID approach. When the new shared frame imported is copies the frame ID and screen buffer reference from old one. But the widget ID become different. So we have IDs inside the screen buffer and in frame itself to be different for same widgets. Moreover we can not even remove old widgets from registry in `LogicalTerminal.replaceFrame()` method because we need the old widgets until the `GenericFrame.importSharedFrame()` finishes to merge visibility state.

Looking for solution. Seems we need to copy ID for every widget from old frame to new one.

**#145 - 09/03/2014 07:26 PM - Eugenie Lyzenko**

My only question is whether the `ScreenBuffer(int frameId, int numWidgets)` constructor is ever safe to use in the new approach? If not, shouldn't we get rid of it?

Not sure, it can be useful sometimes for `ScreenBuffer.getChanged()` method internal call for example.

**#146 - 09/04/2014 07:49 PM - Eugenie Lyzenko**

- File `evl_upd20140904a.zip` added

The today update for review includes:

1. Merge with the recent code base.
2. Fix for shared frame importing `ScreenBuffer` regression.

The idea is to use the widget ID of old frame for widgets in new one. We pass the one more optional parameter to `GenericFrame.coreInitialize()` with array of the widgets from "base" frame and use only ID value from respective widget. Then we register widgets for new frame with old ID in the widget registry. This way we can use `ScreenBuffer` instance from old frame properly. After frame wk is done the old widgets are restored in registry with it's ID. Another word we use same ID with single shared frame for frame and contained widgets. In every shared frame occurrence. Like we use the same `ScreenBuffer` for all shared frames. I did some simple testing and it works. Hope the regression testing will be better now, I'll start it shortly. And in this concept there is no ID leak with shared frames.

**#147 - 09/05/2014 04:48 AM - Vadim Gindin**

Sorry, that I'm meddling. Why your updates does not change `handle.getResourceId` static method (and corresponding `ResourceIdHelper`), that is used to generate IDs for handles resources? I suspected that this algorithm should be replaced with the new one too.

**#148 - 09/05/2014 08:24 AM - Greg Shah**

Code Review 0904a

I'm fine with the changes.

**#149 - 09/05/2014 08:56 AM - Greg Shah**

Sorry, that I'm meddling.

You are not meddling. You are CONTRIBUTING. Your questions help us make this project better, so it is a very good thing.

Why your updates does not change `handle.getResourceId` static method (and corresponding `ResourceIdHelper`), that is used to generate IDs for handles resources? I suspected that this algorithm should be replaced with the new one too.

It is a good question. This is my perspective:

- Handle IDs are made visible to the application code when `STRING(my-handle)` is called. This renders that handle's unique ID as a string. In P2J this is `handle.toString()`.
- You can obtain a handle instance that matches a particular unique ID by using `HANDLE(my-id-representation-as-a-string)`. In P2J this is `handle.fromString()`. Interestingly, you can try passing artificially created strings (in the right format) and if you happen to match a currently existing handle that has that ID, then you will in fact obtain that handle.
- All handles have a unique ID, including non-widget handles like sockets, XML nodes, queries and so forth.
- Widgets are not the same thing as the handle that references them. For example, if you want to pass a reference to a widget as a handle (to a function that is defined like this: `function my-widget-worker returns int (input w as handle): return 0. end.`, you CANNOT use `my-widget-worker(widget-name)`. Instead, you must use `my-widget-worker(widget-name:HANDLE)` which obtains the backing handle for that widget.
- It is true that the 4GL does have an implicit syntax in which you can directly use the widget's name when accessing methods and attributes. This is inconsistent with other handle usage and it makes widgets much more confusing. The 4GL compiler probably just translates such usage into an explicit usage of the associated handle in the "r-code", but it is just not visible to the programmer.
- Although widgets have a `WIDGET-ID` attribute, support for this is only activated when `-usewidgetid` is passed on the 4GL command line as an option. It can only be activated on Windows GUI, ChUI doesn't allow this. When it is activated, it imposes pretty severe limits on the size and behavior of the widget ID pool. If handle IDs and widget IDs were the same thing, then this `-usewidgetid` feature would probably not be needed.
- I believe this code shows that handle IDs are different:

```
def var i as int.  
def frame bogus i.  
view frame bogus.  
message string(i:widget-id in frame bogus) string(i:handle in frame bogus).
```

Run it with:

```
prowin32 -p widget_id_vs_handle_id.p -usewidgetid
```

The resulting message is:

```
65533 1102
```

Because of the above, I think the new widget ID approach should be kept separate from the handle IDs.

**#150 - 09/15/2014 10:11 AM - Constantin Asofiei**

Greg/Eugenie: there are some widgets created on client-side which don't have a server-side counterpart. Looking at the evl\_upd20140904a.zip update, my understanding is that the widget IDs are generated by the server-side and pushed down to the client. My problem is this: all widgets (no matter if there is a server-side counterpart or not) need to have an ID, so that their configuration can be uniquely mapped. To not end up with collisions, we need to either:

1. ensure that the next available ID is pushed from the server-side to client-side and vice-versa (via state sync)
2. assume that i.e. all negative IDs (or a different range) are for widgets which do not have a server-side counterpart
3. something else?

**#151 - 09/15/2014 11:43 AM - Greg Shah**

there are some widgets created on client-side which don't have a server-side counterpart

Please provide more details.

Eugenie handled the tiny input and login client cases already.

ensure that the next available ID is pushed from the server-side to client-side and vice-versa (via state sync)

I like this approach. We will need to send the list of client-side newly allocated IDs AND the list of client-side removed IDs. This is because the server maintains a widget registry and there must be entries in that list for all allocated IDs.

assume that i.e. all negative IDs (or a different range) are for widgets which do not have a server-side counterpart

This makes it harder to implement the "compatible" widget-id support later, so I prefer the sync approach.

**#152 - 09/15/2014 11:49 AM - Constantin Asofiei**

Greg Shah wrote:

there are some widgets created on client-side which don't have a server-side counterpart

Please provide more details.

There are lots of places:

1. label - search for `WidgetFactory.createLabel`
2. button - search for `WidgetFactory.createButton`
3. fill-in - `Browse.startEdit`
4. scrollable-list - `DropDown` c'tor
5. message-area, message-line and status-line pseudo-widgets
6. scrollpane - `SelectionList` c'tor + `Frame` c'tor
7. radio-button - `RadioSet` c'tor
8. the GUI window native widgets correspondents (title-bar, caption buttons, popup menu, icon, etc)

**#153 - 09/15/2014 12:57 PM - Greg Shah**

How many of these cases can act like "real" widgets? In other words, many of these are not treated as widgets by the 4GL. I think 4 through 8 are not real widgets in 4GL terms. Even labels (1) are not really widgets.

For such cases, do we really need widget IDs?

**#154 - 09/15/2014 01:33 PM - Constantin Asofiei**

Greg Shah wrote:

How many of these cases can act like "real" widgets? In other words, many of these are not treated as widgets by the 4GL.

Is not only a question if 4GL maps them as real widgets or internal widgets. My goal is to separate the widget's config (and in the end the runtime state) from its concrete implementation, so the first step was to have an external repository for these config instances. When a widget is created its ID is needed to be able to access this repository: the config is either retrieved from the external repo or a new config instance is created and saved to this repo, based on the id. The config reference saved at a widget instance (server-side or client-side) will always reference the "active" configuration from this repository. So, this will make it easy for different concrete implementations for a widget to share state (i.e. when switching from a GUI to a ChUI mode).

Unfortunately, in lots of cases the Label instance is used by P2J to just compute some metrics - and then completely dropped. I really don't like/want to "hack" the underlying infrastructure just for these cases... I think is better to just properly clean up after it, instead of doing anything else more complex.

I think 4 through 8 are not real widgets in 4GL terms.

I agree they are not, but internally for P2J they are widgets, and some of them need to be part of the official WidgetRegistry. I think we can do this: for the widgets which can work safely without sharing the config in the repository (i.e. its state isn't shared with the server-side, the widget has no correspondent when the streams are switched to ChUI, etc), let the config work as it used to.

Even labels (1) are not really widgets.

There is the SIDE-LABEL-HANDLE attribute for a FILL-IN - so 4GL might have some underlying resource for them. More, if you recall, the frame's header (when the labels are on top) is too split into widgets, in 4GL.

**#155 - 09/15/2014 01:49 PM - Constantin Asofiei**

On a side note, do you now how much work is left on this task? I want to integrate this with my changes.

**#156 - 09/15/2014 02:31 PM - Greg Shah**

My goal is to separate the widget's config (and in the end the runtime state) from its concrete implementation, so the first step was to have an external repository for these config instances.

This makes sense.

Unfortunately, in lots of cases the Label instance is used by P2J to just compute some metrics - and then completely dropped. I really don't like/want to "hack" the underlying infrastructure just for these cases... I think is better to just properly clean up after it, instead of doing anything else more complex.

Agreed. The alternative leaves too many latent problems in our future because some dependencies are implicitly bypassed.

some of them need to be part of the official WidgetRegistry. I think we can do this: for the widgets which can work safely without sharing the config in the repository (i.e. its state isn't shared with the server-side, the widget has no correspondent when the streams are switched to ChUI, etc), let the config work as it used to.

OK.

There is the SIDE-LABEL-HANDLE attribute for a FILL-IN - so 4GL might have some underlying resource for them. More, if you recall, the frame's header (when the labels are on top) is too split into widgets, in 4GL.

Good point.

On a side note, do you now how much work is left on this task? I want to integrate this with my changes.

The evl\_upd20140904a.zip is in regression testing, but Eugenie was not able to finish the testing by the time he had to leave on vacation. He will be back tomorrow and this would be his priority.

This update just puts the new ID infrastructure into place. It doesn't solve any of the other dynamic widget issues. Do you want to integrate these changes and take it forward from here while Eugenie works on the next dynamic widget support?

**#157 - 09/15/2014 02:34 PM - Constantin Asofiei**

Greg Shah wrote:

This update just puts the new ID infrastructure into place. It doesn't solve any of the other dynamic widget issues. Do you want to integrate these changes and take it forward from here while Eugenie works on the next dynamic widget support?

If the update is stable/passed runtime testing I think is best to release this as it is, first. Eugenie then can continue working on the other dynamic widget support stuff, and I will just have to merge my changes with this update.

**#158 - 09/15/2014 02:42 PM - Greg Shah**

OK, we will keep going with the current plan. Hopefully it will pass testing and get checked in tomorrow.

**#159 - 09/17/2014 02:58 PM - Eugenie Lyzenko**

- File *evl\_upd20140917a.zip* added

The regression testing has regression. The next update will be tested as fix. What is in the update:

1. Merge with the recent code base.
2. The `LogicalTerminal.replaceFrame()` is changed to look old frame in `widgetRegistry` instead of `frameRegistry` because we remove frame from `frameRegistry` and there is no `sharedFrameRegistry` where we stored sleeping shared frames and there is no more `activateSharedFrame()` method to move shared frame into `frameRegistry` during `GenericFrame.importSharedFrame()` method.
3. The method `LogicalTerminal.getWidgetForIdInt()` changed to relax LOG writing to \*.log file. If we include -1 into case of logging - the file become very big(~120M) per testing cycle. We have a lot of pseudo-widgets with -1 ID like label widget.

The new testing round will be started shortly.

**#160 - 09/17/2014 07:33 PM - Greg Shah**

Code Review 0917a

I am fine with the changes.

Besides the testing, how soon can you make a list of the changes needed to enable the rest of dynamic widget support?

**#161 - 09/18/2014 08:29 AM - Eugenie Lyzenko**

Now at least two issues/regressions remain:

- one for `NullPointerException`.
- other with shared variable missing.

So I need to debug them.

Besides the testing, how soon can you make a list of the changes needed to enable the rest of dynamic widget support?

I had a prototype that worked before I started to work new ID implementation. So the changes can look like this:

1. Modify `DynamicWidgetFactory.create()`:

```
public static void create(String widgetType, handle h, character widgetPool)
{
...
    try
    {
        Constructor<? extends GenericWidget> ctor = widCls.getDeclaredConstructor(boolean.class);
        WrappedResource res = ctor.newInstance(true);

        h.assign(res);

        int newId = WidgetId.allocateId();
        h.unwrapWidget().setId(newId);

        WidgetPool.addResource(res, widgetPool);
    }
...
}
```

2. Modify `GenericWidget.setFrame()` method to notify frame about requirement to add dynamic widget:

```
...
public void setFrame(GenericFrame frame)
{
    this.frame = frame;

    // dor dynamic widgets this is the place where we need to initialize all frame internals
    // related to the wiget integration into the frame functionality
    if (dynamic)
    {
        frame.addDynamicWidget(this);
    }
}
...
```

3. Add respective calls to `GenericFrame` to handle dynamic widget creation/delete:

```
...
/**
 * Adding new dynamic widget to the current frame.
 *
 * @param dynWidget The widget to be added to current frame.
 */
void addDynamicWidget(GenericWidget dynWidget)
{
    // adding new widget to the name map
    n2w.put(getDynamicWidgetName(dynWidget), dynWidget);

    // add new widget to the widgets array of the current frame
    GenericWidget[] newWidgets = new GenericWidget[n2w.size()];
    for (int i = 0; i < widgets.length; i++)
    {
        newWidgets[i] = widgets[i];
    }
    newWidgets[widgets.length] = dynWidget;
    widgets = newWidgets;

    // refresh frame definition if exists
    if (frameDef != null)
    {
        frameDef.addConfig(dynWidget.getConfig());
    }
}
```

```

    }
}

/**
 * Removing dynamic widget from the current frame.
 *
 * @param    dynWidget  The widget to be deleted from the current frame.
 */
void deleteDynamicWidget(GenericWidget dynWidget)
{
    // hide from the screen
    dynWidget.hide();

    // removing widget from name map
    n2w.remove(getDynamicWidgetName(dynWidget));

    // remove new widget from the widgets array of the current frame
    GenericWidget[] newWidgets = new GenericWidget[n2w.size()];
    for (int i = 0; i < widgets.length; i++)
    {
        if (widgets[i].getId() != dynWidget.getId())
        {
            newWidgets[i] = widgets[i];
        }
    }
    widgets = newWidgets;

    // refresh frame definition if exists
    if (frameDef != null)
    {
        frameDef.removeConfig(dynWidget.getConfig());
    }
    pushScreenDefinition();
}

/**
 * Construct the string name for dynamic widget based on unique ID number.
 *
 * @param    dynWidget  The widget the name will be generated to.
 *
 * @return    The String for dynamic widget name in form "dynWid"+"ID".
 */
private String getDynamicWidgetName(GenericWidget dynWidget)
{
    StringBuffer sb = new StringBuffer("dynWid").append(dynWidget.getId());

    return sb.toString();
}
...

```

4. The call `GenericWidget.resourceDelete()` to remove widget:

```

...
protected boolean resourceDelete()
{
    if (!canDelete())
    {
        return false;
    }

    // a static widget can be deleted ONLY if the parent frame is not visible,
    // forced or not
    if (!_dynamic() && frame != null && frame.asWidget().getConfig().isVisible)
    {
        return false;
    }

    // TODO: remove the widget, etc
    // removing dynamic widget from associated frame
    if (_dynamic() && frame != null)
    {

```

```
        frame.deleteDynamicWidget(this);
    }
    deleted = true;

    return true;
}
...
```

Some notes:

I used `GenericFrame.getDynamicWidgetName()` to be able to add widget to map having unique name for every widget. This is fast solution, may be we need something smarter.

We will have to think about `widgets[]` array in `GenericFrame` class. We need dynamic array and reallocate full every time we add the widget is too expensive. May be `ArrayList` is a replacement.

#### #162 - 09/18/2014 09:46 AM - Greg Shah

I am fine with the proposed approach. Some thoughts:

1. The use of reflection (to lookup the constructor) needs some safety code added to handle all possible failure conditions (exceptions and NPE).
2. In regards to the dynamic widget name, I think you can just use the ID as a string. We don't need the "dynWid" prefix since the ID itself cannot ever be a valid static widget name AND we know that the ID is unique.
3. I think it is OK to reallocate the widget array when a dynamic widget is added or removed. Such add/remove operations happen infrequently compared to the number of times we access the widget array in normal use. It is best to leave the widget array as is instead of moving to `ArrayList`. Otherwise we would have to use `ArrayList.toArray()` in the common use case when we would otherwise just be able to access the array directly.

Get this update prepared and uploaded for review ASAP. Please also test it with a variety of dynamic widget cases to see if there are any other problems.

#### #163 - 09/18/2014 10:39 AM - Eugenie Lyzenko

Get this update prepared and uploaded for review ASAP. Please also test it with a variety of dynamic widget cases to see if there are any other problems.

This should be done with the code for new widget ID implementation?

**#164 - 09/18/2014 10:41 AM - Greg Shah**

Yes, it must build upon (add to) that code.

**#165 - 09/18/2014 10:43 AM - Greg Shah**

Your top priority is getting the new widget ID implementation through regression testing and checked in.

However, while you are waiting for tests to run, I want you to make headway on the rest of the dynamic widget work. I'd like to get the ID stuff done this week and the dynamic widget support finished as early next week as possible. Many people's tasks are dependent upon this work.

**#166 - 09/18/2014 11:57 AM - Eugenie Lyzenko**

- File *evl\_upd20140918a.zip* added

Your top priority is getting the new widget ID implementation through regression testing and checked in.

OK.

However, while you are waiting for tests to run, I want you to make headway on the rest of the dynamic widget work. I'd like to get the ID stuff done this week and the dynamic widget support finished as early next week as possible. Many people's tasks are dependent upon this work.

I see. Fortunately these task are pretty independent so we can do this in simultaneously.

This update for review includes:

1. Dynamic widget support suggested approach. Forgot to note about ScreenDefinition class change. We need to add method to remove dynamic widget config when widget is gone.
2. Fix for NPE in regression testing. Event we release shared frame we need to keep widgets in registry because they can be used further.

The new testing cycle will be started shortly after DB is restored.

**#167 - 09/18/2014 12:32 PM - Greg Shah**

Code Review 0918a

I only have the following concerns:

1. The changes to LT bring back that memory leak of shared frames, right? I understand it may be needed to defer the cleanup longer than we would like. The problem: can we find the right place to do the final cleanup? I wonder if we can do the removal when exiting the external procedure containing the earliest reference to a given shared frame.

2. Can the TODO from `GenericWidget.resourceDelete()` be removed now?

**#168 - 09/18/2014 01:30 PM - Eugenie Lyzenko**

2. Can the TODO from `GenericWidget.resourceDelete()` be removed now?

I think we can remove. This is not my TODO but I think if we need more cleanup - it will be found in regression testing.

**#169 - 09/18/2014 01:53 PM - Eugenie Lyzenko**

1. The changes to LT bring back that memory leak of shared frames, right? I understand it may be needed to defer the cleanup longer than we would like. The problem: can we find the right place to do the final cleanup? I wonder if we can do the removal when exiting the external procedure containing the earliest reference to a given shared frame.

The method I've modified is: `LT.releaseSharedFrame()`. It is called only from `GenericFrame.createSharedFrame()` when the new shared frame is created and previously we already had the frame with same name and same ID. I think we can suppress content cleanup because it will be replaced further by created frame content. So may be this is not a memory leak.

On the other hand I think the shared frame in 4GL approach has memory leak bomb initially. We never know at this time if the frame will be used anymore or not. So once the shared frame is creating we need to keep it until application ends. Correct me if I'm wrong.

**#170 - 09/18/2014 05:01 PM - Greg Shah**

On the other hand I think the shared frame in 4GL approach has memory leak bomb initially. We never know at this time if the frame will be used anymore or not. So once the shared frame is creating we need to keep it until application ends.

No, there is no such thing as a NEW GLOBAL SHARED frame. You can only create a scoped shared (NEW SHARED) frame which can only be accessed as long as the external procedure that created it exists. When not using persistent procedures, this means that when the external procedure exits, all shared frames created by it should be removed.

This memory leak already exists in P2J. So we can go forward with your change anyway. But I do want to fix this sometime soon.

**#171 - 09/19/2014 06:24 AM - Eugenie Lyzenko**

This memory leak already exists in P2J. So we can go forward with your change anyway. But I do want to fix this sometime soon

OK.

Good news. The main regression testing part completed without any regression(only one failed test TC-JOB-002 with known EOF). Starting the CTRL-C part.

**#172 - 09/19/2014 09:57 AM - Eugenie Lyzenko**

- File *evl\_upd20140919a.zip* added

The only change in this update is removed TODO from GenericWidget.resourceDelete().

Is it OK to commit and distribute this one if CTRL-C tests are passing without regression?

**#173 - 09/19/2014 11:05 AM - Eugenie Lyzenko**

The CTRL-C testing finished without regressions. I had to run 3-way tests in standalone session and they are OK too.

So I'm going to commit and distribute changes if you do not mind.

**#174 - 09/19/2014 11:06 AM - Greg Shah**

Let me take one last look first.

**#175 - 09/19/2014 11:16 AM - Eugenie Lyzenko**

Let me take one last look first.

OK.

No, there is no such thing as a NEW GLOBAL SHARED frame. You can only create a scoped shared (NEW SHARED) frame which can only be accessed as long as the external procedure that created it exists. When not using persistent procedures, this means that when the external procedure exits, all shared frames created by it should be removed.

May be we need to have some kind of "exit processing" class/method. When the procedure enter into own scope we create list of shared frames that is filling with frames created inside procedure. So when we leave the procedure scope - the exit handler will remove all shared frames created in this scope.

**#176 - 09/19/2014 11:32 AM - Greg Shah**

OK, go ahead and commit/distribute evl\_upd20140919a.zip.

No, there is no such thing as a NEW GLOBAL SHARED frame. You can only create a scoped shared (NEW SHARED) frame which can only be accessed as long as the external procedure that created it exists. When not using persistent procedures, this means that when the external proc exits, all shared frames created by it should be removed.

Maybe we need to have some kind of "exit processing" class/method. When the procedure enters into its own scope we create a list of shared frames that is filled with frames created inside the procedure. So when we leave the procedure scope - the exit handler will remove all shared frames created in this scope.

Yes, we can leverage support that we already have in the TransactionManager for exit processing. I have created another task for that (see [#2396](#)), since the leak is already there today. You don't have to work on that further.

Please move forward with testing/finishing the dynamic widget support ASAP. One area I wonder about is whether the client-side WidgetRegistry needs to add/remove dynamic widgets differently.

LE: EVL checked this into bzt rev 10615.

**#177 - 09/19/2014 01:37 PM - Eugenie Lyzenko**

One area I wonder about is whether the client-side WidgetRegistry needs to add/remove dynamic widgets differently.

Some thoughts about this.

1. After the dynamic widget is created and attached to the frame (assigning frame attribute) the widget becomes a regular one from the client perspective.
2. Client takes screen definition with static and dynamic widgets. All of them become the part of the WidgetRegistry. I think there is no difference between static or dynamic widgets.
3. When the frame is destroyed - all widgets inside are removed from the client's WidgetRegistry.

If 1-3 will work fine I think we do not need any special processing for dynamic widgets on the client side. What do you think?

**#178 - 09/19/2014 02:43 PM - Greg Shah**

If 1-3 will work fine I think we do not need any special processing for dynamic widgets on the client side. What do you think?

If the client can't ever work with widget that is not part of a frame, then you are probably right. We will see.

What is important for now is for you to create and execute a variety of dynamic widget testcases to see if there is some core dynamic widget behavior that we don't support yet. Obviously, you must stay within the attributes/methods and widget types that we already support. That means testing only on ChUI right now.

For now we also can't really test multiple windows. So please focus on testing a wide range of ChUI default-window dynamic widget support. Test:

- all of the widget types that we support (for ChUI I think that means anything except menus)
- frames with all dynamic widgets
- frames with a mixture of dynamic and static widgets
- dynamic frame creation
- use of the above different frame types for redirected terminal, for UPDATE/SET/PROMPT-FOR, for ENABLE/WAIT-FOR, for CHOOSE, DISPLAY
- test firing triggers on dynamic widgets
- test validation expressions on dynamic widgets

Hopefully you get the idea. Make sure that our support works well.

**#179 - 09/23/2014 01:41 PM - Greg Shah**

How much testing have you done and what are your findings?

**#180 - 09/23/2014 03:02 PM - Eugenie Lyzenko**

There is an issues creating something other than fill-in widget. Consider the following code in DynamicWidgetFactory:

```
...
static
{
    widgetFactory.put (LegacyResource.BUTTON,      ButtonWidget.class);
    widgetFactory.put (LegacyResource.COMBO_BOX,    ComboBoxWidget.class);
    widgetFactory.put (LegacyResource.CONTROL_FRAME, null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.DIALOG_BOX,   null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.EDITOR,       EditorWidget.class);
    widgetFactory.put (LegacyResource.FILL_IN,       FillInWidget.class);
    widgetFactory.put (LegacyResource.FRAME,         FrameWidget.class);
    widgetFactory.put (LegacyResource.IMAGE,         null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.MENU,          null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.MENU_ITEM,     null); // TODO: what is the widget ?
}
```

```

widgetFactory.put (LegacyResource.RADIO_SET,      RadioSetWidget.class);
widgetFactory.put (LegacyResource.RECTANGLE,      RectangleWidget.class);
widgetFactory.put (LegacyResource.SELECTION_LIST, SelectionListWidget.class);
widgetFactory.put (LegacyResource.SLIDER,         null); // TODO: what is the widget ?
widgetFactory.put (LegacyResource.SUB_MENU,       null); // TODO: what is the widget ?
widgetFactory.put (LegacyResource.TEXT,           TextWidget.class); // TODO: what is the widget ?
widgetFactory.put (LegacyResource.TOGGLE_BOX,     ToggleBoxWidget.class);
widgetFactory.put (LegacyResource.WINDOW,         WindowWidget.class);
}
...

```

Does it mean the following widgets are not supported at this time:

```

...
    widgetFactory.put (LegacyResource.CONTROL_FRAME, null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.DIALOG_BOX,   null); // TODO: what is the widget ?
...
    widgetFactory.put (LegacyResource.IMAGE,        null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.MENU,         null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.MENU_ITEM,    null); // TODO: what is the widget ?
...
    widgetFactory.put (LegacyResource.SLIDER,       null); // TODO: what is the widget ?
    widgetFactory.put (LegacyResource.SUB_MENU,     null); // TODO: what is the widget ?
...

```

Or we plan to implement the support now?

#### #181 - 09/23/2014 03:23 PM - Constantin Asofiei

Eugenie, you can ignore these for now. Your support for dynamic widget creation should be generic enough to be able to add these too easily, in the future.

#### #182 - 09/24/2014 01:11 PM - Eugenie Lyzenko

From the testing I did I can tell we certainly need to change the ScreenBuffer class. To be able to extend the content of the widgets the frame has. Every new dynamic widget should have at least the state representation in screen buffer.

My plan is to implement the dynamic widgets to be the regular ones from the moment the widget is adding to the frame. It should simplify the

debugging and life cycle tracking.

Another point is to give the server part all widget life aspect control. Meaning we will create and delete dynamic widgets on server side. Then widget is attached to the frame on server side and acts as regular widget from the client perspective.

The dynamic widget specifics(like warning messages for incorrect label setting for example) can also be done on the server side.

The first look shows the problems with dynamic frame creation. The trigger for dynamic widgets also does not work as expected(or just does not work).

**#183 - 09/24/2014 04:41 PM - Greg Shah**

OK, this all sounds like a reasonable approach. How quickly can you implement the necessary changes?

**#184 - 09/24/2014 05:13 PM - Eugenie Lyzenko**

OK, this all sounds like a reasonable approach. How quickly can you implement the necessary changes?

I guess if there will be no any hidden issues I do not know at this time I need 1-2 days to have code to run regression testing. Anyway I try to complete ASAP. The areas of work:

- create/delete all widgets known to the DynamicWidgetFactory
- debug/implement widget event processing
- debug/implement error warnings for attributes that can not be set for dynamics
- verify batch mode and mix for dynamic and static widgets

If something is not supported on conversion level(like RADIO-BUTTONS attribute for RADIO-SET widget) I can postpone implementation. Correct?

**#185 - 09/24/2014 05:34 PM - Greg Shah**

If something is not supported on conversion level(like RADIO-BUTTONS attribute for RADIO-SET widget) I can postpone implementation. Correct?

Yes.

**#186 - 09/26/2014 07:52 AM - Eugenie Lyzenko**

Triggering issue to discuss.

The problem I'm currently work with is related to the triggers identification for dynamic widgets. The trigger registration code converted and put into main body in a time when the dynamic widget is not yet created. All we have at this time is prepared handle object without attached WrappedResource object. So we allocate resource ID for null value. handle class put the mapping null->longIdNumber into idResources, resourceIds maps. And moreover this ID is the same for all dynamic widget triggers registered this way.

So the triggers should be attached to the handle itself, not to the widget object(which might be not yet created). Then after event is happening we need to identify correct handle attached and correct wrapped resource(trigger source).

**#187 - 09/26/2014 08:55 AM - Greg Shah**

So the triggers should be attached to the handle itself, not to the widget object(which might be not yet created). Then after event is happening we need to identify correct handle attached and correct wrapped resource(trigger source).

We don't want to change how the client works. The client has no knowledge of handles. And to know when to fire a trigger, you must know the widget upon which the event is occurring.

But the good news is that at the time of the WAIT-FOR, when we gather up the event-lists on the server side, that is the time that any event target that is a dynamic widget must already have been created, attached to a frame and enabled for input. This means that we just have to defer the resolution of the widget from the handle, until the time we are sending the event-lists to the client. Please see LogicalTerminal.prepareEventList() and TriggerManager.mergeTriggerRegistry().

By resolving the handle to widget mapping at the server, the client will be unchanged.

**#188 - 09/26/2014 01:16 PM - Eugenie Lyzenko**

We don't want to change how the client works. The client has no knowledge of handles. And to know when to fire a trigger, you must know the widget upon which the event is occurring.

Agreed with this approach. From the client view there should be no differences between static and dynamic widgets.

More findings for triggers. The triggers works fine when dynamic widgets are creating on application startup(before the initial event list are constructing).

But imagine the case when dynamic widget creation is the result of user(or application logic) intervention. For example we have a button to create dynamic widgets. So I think this is the case when we get WAIT-FOR but still do not have the dynamic widgets. But the event list already done for null resource handles. This is the case I'm trying to resolve.

But imagine the case when dynamic widget creation is the result of user(or application logic) intervention. For example we have a button to create dynamic widgets. So I think this is the case when we get WAIT-FOR but still do not have the dynamic widgets. But the event list already done for null resource handles. This is the case I'm trying to resolve.

Hmmm. Yes, I see your point here. But I think there can't be new triggers added without a new WAIT-FOR. In other words, in the 4GL triggers are something that are new blocks of code that have a scope. And you cannot define a new trigger from inside another trigger. To create a new trigger you must execute an internal or external procedure. You can call such a thing from a trigger BUT while it is executing the client is NOT responsive to user input. So any added triggers only become active if there is another WAIT-FOR (a nested one in this case).

For this reason, I don't think that a dynamically created button can be the direct target of an existing trigger. However, it could be the target of an ANYWHERE trigger. The question is: does our current client code properly handle this case? The trigger would already be existing (ON "CHOOSE" ANYWHERE).

Am I incorrect in my assessment?

#190 - 09/26/2014 02:33 PM - Constantin Asofiei

Greg/Eugenie: first of all, a trigger can't be registered with a handle which is not valid. The widget (dynamic or not) must already exist when the trigger is defined. But this doesn't prohibit for the trigger to be registered with a widget which doesn't belong to a frame. Consider this code:

```
def var ch as char.
def var h as handle.

create button h.
h:label = "aaa".
h:row = 4.

on any-key of h do:
    message h:type.
end.

form ch with frame f1 size 20 by 10.

on 1 anywhere do:
    h:frame = frame f1:handle.
    view frame f1.
    enable all with frame f1.
end.

wait-for close of current-window.
```

There is only one WAIT-FOR loop. Pressing 1, it will attach the BUTTON represented by handle h to frame f1. Enabling all widgets in frame f1 will allow you to press i.e. space on the button - the trigger on the button will be seen.

Second case: if the widget referenced by the handle is changed by the trigger, i.e.:

```

on 1 anywhere do:
  create button h.
  h:label = "bbb".
  h:row = 4.
  h:frame = frame f1:handle.
  view frame f1.
  enable all with frame f1.
end.

```

The trigger defined on the button with label aaa will no longer be seen. I think the conclusion is that the triggers can be defined only for existing widgets (dynamic or not, attached to a frame or not).

#### #191 - 09/27/2014 12:24 PM - Eugenie Lyzenko

- File *evl\_upd20140927a.zip* added

This is the main dynamic widget support update for review. Has the incorrect attributes handling, dynamic frames, widgets defined in DynamicWidgetFactory, Triggers are verified to work as expected. In 4GL if no widget yet created for handle - no trigger available to attach.

The dynamic widgets are now added to the ScreenBuffer of the respective parent frame. The dynamic frames are creating by GenericFrame from scratch with new interface DynamicFrameTemplate. So we have new GenericFrame instance for every new dynamic frame.

The base approach logic is there however some changes are possible during review/debug/test. So if you do not mind I'll start the regression testing tonight.

As a general note some widgets functionality can conflict with dynamic approach(selection list is an example). But I guess this is not pure dynamic issues but may be hidden widget problems/bugs we did not encounter yet. So if there are no approach objection I think we can commit the changes as the base after testing is done without regressions.

#### #192 - 09/28/2014 09:55 AM - Greg Shah

Code Review 0927a

It looks good! As you suggest, if it passes testing you can check it in since any broken dynamic features won't cause a problem to MAJIC. However, I do want to resolve these things. They can be fixed now if regression testing failed. Or they can be fixed in the next pass if regression testing passed.

1. Put a comment in LogicalTerminal.addDynamicFrame() that the frame will be removed in FrameWidget.resourceDelete().
2. Why aren't dynamic frames added to the widget pool in DynamicWidgetFactory.create()?
3. I'm not sure opening the scope of a dynamic frame in GenericWidget.setId() is correct. That will associate it with the current block which can be a nested block instead of the external procedure. I suspect that any dynamic frame acts like it is scoped to the same block as its widget pool or possible as its containing external procedure. Please try to test this to confirm.
4. The GenericFrame changes could be done with common code in worker methods and an additional method with an extra boolean dynamic parameter. That would reduce some code duplication.

**#193 - 09/28/2014 02:34 PM - Eugenie Lyzenko**

The changes passed the regression testing.

3. I'm not sure opening the scope of a dynamic frame in `GenericWidget.setId()` is correct. That will associate it with the current block which can be a nested block instead of the external procedure. I suspect that any dynamic frame acts like it is scoped to the same block as its widget pool or possible as its containing external procedure. Please try to test this to confirm.

I just thought we need to open the scope somewhere. If it would be done without in `openScope()` within `GenericWidget.setId()` - it is better to remove. I'll remove it in today update later. The question here if the dynamic frame is the child of the frame where it created or main window? What do you think, looks like our frames can not have the children frames so all frames, dynamic or static have the same parent - session window. Correct?

**#194 - 09/28/2014 04:28 PM - Eugenie Lyzenko**

- File `evl_upd20140928a.zip` added

This update for review reflects your recent notes. I'm going to put it on regression testing cycle shortly.

**#195 - 09/29/2014 05:15 AM - Eugenie Lyzenko**

- File `evl_upd20140929a.zip` added

Small change to fix resource type requesting in `GenericWidget` for building the error message. The testing cycle is now restarting for this update.

**#196 - 09/29/2014 08:40 AM - Greg Shah**

3. I'm not sure opening the scope of a dynamic frame in `GenericWidget.setId()` is correct. That will associate it with the current block which can be a nested block instead of the external procedure. I suspect that any dynamic frame acts like it is scoped to the same block as its widget pool or possible as its containing external procedure. Please try to test this to confirm.

I just thought we need to open the scope somewhere. If it would be done without in `openScope()` within `GenericWidget.setId()` - it is better to remove. I'll remove it in today update later.

I do think we need to open the scope. I just think that the approach will have to be different, because our usual way we open the scope "attaches" itself to the current block. I really doubt that is right.

It is important to write 4GL testcases to determine the right behavior. We should not waste time guessing at what is right. Prove it with testcases. There are questions to answer:

- Does a dynamic frame scope itself to a block? If so, then which block (containing external proc, nearest top-level block, nearest enclosing block that has certain properties)?
- Does a dynamic frame scope itself using the widget pool?

Constantin: any thoughts?

The question here if the dynamic frame is the child of the frame where it created or main window?

Do NOT guess. Write testcases to find out how the 4GL works.

What do you think, looks like our frames can not have the children frames so all frames, dynamic or static have the same parent - session window. Correct?

For now, we don't support frame families. But we can document the behavior that is implemented by the 4GL, so that we know what to do later. Please document your results in [#1798](#).

#### **#197 - 09/29/2014 09:03 AM - Greg Shah**

Code Review 0929a

The code looks good.

The last problem is the `openScope()`. We definitely need to execute the `openScope()` (or at least most of it), but we need to know how to do that properly. Please see the `GenericFrame.iterate()`, `GenericFrame.retry()` and `GenericFrame.finished()` to see all the logic that gets executed based on block processing events. Without the `TransactionManager.registerFinalizable()`, these don't occur. But it may be that we need to register it differently.

Without the `openScope()`, the `LogicalTerminal` registrations and frame state will be wrong too.

In writing testcases for frame scoping, you may be able to determine the behavior as follows:

1. In procedure X, create a named widget pool. Then call procedure Y.
2. In procedure Y, execute `CREATE FRAME` in a nested `REPEAT` block (use `LEAVE` to make sure the `REPEAT` doesn't loop). `REPEAT` blocks naturally have scoping/transaction/error properties so if the scope attaches to the `REPEAT`, then you should know it. Make sure the `CREATE FRAME` uses the named widget pool from procedure X.
3. Try to use the frame in procedure Y, but outside (after) the `REPEAT` block. If that works, then you know that the scope is not at the `REPEAT` block level. Now, let procedure Y return.
4. Try to use the frame in procedure X, after procedure Y returns. If that works, then you know that the scope is not attached to the external procedure Y.
5. Delete the named widget pool.
6. Try to use the frame again. This should fail if the frame has been deleted.

#### **#198 - 09/29/2014 04:58 PM - Eugenie Lyzenko**

3. Try to use the frame in procedure Y, but outside (after) the `REPEAT` block. If that works, then you know that the scope is not at the `REPEAT` block level. Now, let procedure Y return.

The answer is positive, the scope is not at the REPEAT block level

4. Try to use the frame in procedure X, after procedure Y returns. If that works, then you know that the scope is not attached to the external procedure Y.

The answer is positive. Frame handle valid(but frame already not visible at this time). So not attached to external procedure.

5. Delete the named widget pool.

6. Try to use the frame again.

This fails. Application terminates. Frame invalid.

So I think the conclusion is scope is attached to widget pool, correct?

**#199 - 09/29/2014 05:07 PM - Greg Shah**

So I think the conclusion is scope is attached to widget pool, correct?

It tells us that at least some of the scoping behavior (frame deletion) is deferred to be processed when the widget-pool is deleted.

However, there is other behavior that may still be scoped to the containing block since leaving that block may have hidden the frame. I think we will need more testcases to fully figure out the behavior.

Please post the testcase code here.

How did your regression testing go? Is it complete? If it passes, please do check in your code and distribute it.

## #200 - 09/29/2014 05:08 PM - Greg Shah

In other words, we will deal with the dynamic frame scoping as the next update (after you get back).

## #201 - 09/29/2014 06:22 PM - Eugenie Lyzenko

Please post the testcase code here.

Here it is. Pretty ugly but I guess do work.

```
=====
DEFINE VARIABLE frh as widget-handle.
def var bh1 as widget-handle.
def var bh2 as widget-handle.

run xx.

procedure xx:

create widget-pool "fr_pool".

message "press key to start external.".
PAUSE.

run yy.

message "press key to use frame outside ext proc.".
pause .

create button bh1 assign
    frame = frh
    label = "Dynamic button xx1"
    sensitive = true
    visible = true
    row = 3
    column = 4.

delete widget-pool "fr_pool".

message "press key to use frame after widget pool delete.".
pause .

create button bh2 assign
    frame = frh
    label = "Dynamic button xx2"
    sensitive = true
    visible = true
    row = 3
    column = 24.

end procedure.

procedure yy:

def var i as int initial 1.
def var bh0 as widget-handle.

repeat STOP-AFTER 1:

    create frame frh in widget-pool "fr_pool" assign
        ROW = 3
        COLUMN = 1
        width = 79
        HEIGHT = 15
        SENSITIVE = TRUE
        VISIBLE = TRUE.

    i = i + 1.
```

```

    IF i > 1 THEN LEAVE.

end.

message "internal proc, frame".
pause.

create button bh0 assign
    frame = frh
    label = "Dynamic button yy"
    sensitive = true
    visible = true
    row = 5
    column = 4.

pause.

end procedure.
=====

```

How did your regression testing go? Is it complete? If it passes, please do check in your code and distribute it.

The main part finished without regressions, waiting for CTRL-C part to be done. I do not expect any issues, so I think today later I'll distribute the changes(0929a).

#### #202 - 09/29/2014 07:52 PM - Eugenie Lyzenko

The CTRL-C part completed without regressions. Had to start 3-way tests in separate session to get all passed in sum. Will commit shortly.

#### #203 - 09/29/2014 08:00 PM - Eugenie Lyzenko

Committed to bzs as 10617.

#### #204 - 09/30/2014 08:14 AM - Greg Shah

From EVL via email:

Just for you to know: in addition there are some missing functionality on client side I've discovered today debugging this approach(like TODO in ThinClient for setLocation() method for example).

**#205 - 10/09/2014 04:38 PM - Eugenie Lyzenko**

Investigating the scoping approach for dynamic widgets I've found interesting info in Progress Programming Handbook for 9.1 page 20-2 - 20-14:

...

Note that dynamic widgets do not adhere to scoping rules. That is, they exist until you delete them or the session ends. Thus, if you create a dynamic button in the FOR EACH loop of a subprocedure, that button continues to exist after the subprocedure returns. However, note that any widget handle variables are locally or globally scoped as you define them. Also, while dynamic widgets are not scoped, trigger definitions are scoped. Thus, for dynamic widgets, define persistent triggers or put your trigger definitions in a persistent procedure.

...

The theory I'm going to check is: we need to keep dynamic frame scope within Dynamic Widget Pool(named or unnamed) that holds the dynamic widget. The features like visibility and trigger attachment will be handles specifically for dynamic frames because they have specific mode for this case.

The testcase I used confirms this theory but I'm going to write more tests, trying to decline it.

**#206 - 10/10/2014 09:21 AM - Eugenie Lyzenko**

- File *evl\_upd20141010a.zip* added

Small drop to fix the missed/not required history entries. Will be committed and distributed soon.

**#207 - 10/10/2014 09:22 AM - Eugenie Lyzenko**

Committed in bzs as 10624. No testing required.

**#208 - 10/10/2014 12:12 PM - Greg Shah**

Investigating the scoping approach for dynamic widgets I've found interesting info in Progress Programming Handbook for 9.1 page 20-2 - 20-14:

Good information. This is what I expected.

The theory I'm going to check is: we need to keep dynamic frame scope within Dynamic Widget Pool(named or unnamed) that holds the dynamic widget.

As far as I know, there is no such thing as a "dynamic widget pool". We already have full widget-pool support and I think that should be enough for

the dynamic widget case. Let's just use the current widget pool support unless you find proof that it is not sufficient.

The features like visibility and trigger attachment will be handles specifically for dynamic frames because they have specific mode for this case.

Please look carefully at all of the frame features that are called by scoping. Look at the following: `GenericFrame.openScope()`, `GenericFrame.iterate()`, `GenericFrame.retry()` and `GenericFrame.finished()`. Please also look at all the behavior (client and server) that is caused by the `LogicalTerminal.signalFrameScope()` processing. For each of the features being implemented based on these "hooks", try to write testcases to see which of these are needed for dynamic frames. I think much of it is not needed. For example, I suspect that dynamic frames have not automatic DOWN processing on `iterate()` and they don't reset on `retry()`. However, there may be some things that are needed like the processing of implicit HIDE.

Document your findings here and check in your testcases to a `testcases/uast/dynamic_frame_scoping/` directory.

Constantin: do you have anything to add?

#### #209 - 10/10/2014 12:36 PM - Constantin Asofiei

Greg Shah wrote:

Constantin: do you have anything to add?

Dynamic widgets can survive the creator procedure, if the procedure is ran persistent. Make sure to test something like this:

```
def new shared var hframe as handle.
def var hp as handle.

run <some-procedure-which-creates-dynamic-frame>.p persistent set hp.

/* use hframe here - it should work */

delete object hp.

/*
after the instantiating procedure has been deleted, the frame should no longer be valid, and use in VIEW/DISPLAY should no longer be possible.
but frame visibility state might affect this (i.e. frame's handle is not deleted until the frame is hidden).
*/
```

This is similar with the buffer's scoping findings: all dynamic buffers are added to a global scope, where they are alive until they are explicitly deleted or their instantiating procedure is deleted. Dynamic frames might have some similar behaviour.

**#210 - 10/13/2014 04:06 PM - Eugenie Lyzenko**

- File testcases\_ev/20141013a.zip added

For iterate() hook. As we expected from the 4GL documentation the dynamic frame is only one down frame. Trying to use iteration produces the error on compile/syntax check time(dyn\_wid\_test20.p). The message is:  
"Static frame name fdyn conflicts with widget-handle variable. (3517)"

For retry() hook. The dynamic frame does not reset on retry. I have written the testcase similar to existed frame\_retry.p. The testcase dyn\_wid\_test21.p shows the current values in the dynamic frame is not changing. So I guess we can safely ignore this functionality in dynamic frame case.

For finished hook. Actually I think we need some kind of finalization for dynamic frames. Just because any pending actions waiting to be completed should be completed when the frame terminating. For example we need to close and write any opened streams. I'll try to write the appropriate testcase to check this. Continue working.

**#211 - 10/13/2014 07:08 PM - Greg Shah**

As we expected from the 4GL documentation the dynamic frame is only one down frame.

One thing that is strange is that you can set dynframe:DOWN = 0. which normally means "this is a down frame with the number of lines determined by the available space in the window at runtime". It is possible that this is just ignored, but I am worried that there is some way to make a dynamic frame into a DOWN frame.

I do agree that I can't yet think of a way to make static frame statements like DISPLAY or DOWN work with dynamic frames.

Constantin: do you have any ideas here?

For retry() hook. The dynamic frame does not reset on retry. I have written the testcase similar to existed frame\_retry.p.

Where is frame\_retry.p? I don't see it in testcases/uast/.

I don't see how dyn\_wid\_test21.p shows a retry. Does message retry. ever report yes during that code? Where is the error that causes the retry to occur?

Greg Shah wrote:

I do agree that I can't yet think of a way to make static frame statements like DISPLAY or DOWN work with dynamic frames.

Constantin: do you have any ideas here?

I can't find anything on the web related to dynamic frames to be used in stmts which refer to static frames. All I found was a CLEAR ALL reference: <http://knowledgebase.progress.com/articles/Article/P106792> which states that CLEAR ALL works poorly with dynamic widgets.

I didn't look over all of Eugenie's tests, but I think this applies:

1. if a dynamic frame is created dynamically, setting its DOWN to 0 will automatically compute the down relative to the window's height, when the frame is realized, i.e.:

```
def var h as handle.  
  create frame h.  
  h:down = 0.  
  view h.  
  message h:down.
```

But this will not affect the frame's HEIGHT-CHARS attribute: this still needs to be manually specified.

2. the dynamic widgets in a dynamic frame do not get duplicated for each line in the frame's down body.
3. if the frame is static, with down set to 0 and it contains both dynamic and static widgets, the static widgets will get duplicated in each line of the frame's down body.

```
def frame f1 with down width-chars 40 side-labels.  
  
  def var h as handle.  
    create fill-in h.  
    h:data-type = "integer".  
    h:screen-value = "10".  
    h:format = "999,999". /* this ensures if the down body contains it, it will not show an empty space */  
    h:frame = frame f1:handle.  
    h:row = 1.  
    h:column = 15.  
    h:visible = true.  
  
    def var i as int.  
      do i = 1 to 10:  
        display i with frame f1.  
        down with frame f1.  
      end.
```

4. a static widget can't change its frame: thus it can't be added to a dynamic frame: \*\*FRAME is an invalid attribute for a static FILL-IN i. (4071)

```
def var h as handle.  
  
def var i as int.  
  
display i with 10 down frame f1.  
  
create frame h.  
h:down = 0.  
i:frame = h.  
  
view h.
```

The conclusion I think is that stmts which work with static frame/widgets can't work with dynamic frame/widgets.

**#213 - 10/14/2014 10:44 AM - Greg Shah**

Eugenie:

1. Please ensure that all 3 examples shown in note 212 are checked in to the dynamic widget testcases and are properly handled in P2J.
2. Try using a 0 down dynamic frame inside of a REPEAT loop. Make sure all access to that frame (except for the CREATE) is inside that loop. After iteration, can you detect any features of GenericFrame.iterate()? For example, does the equivalent of resetFrameBuf() get executed?
3. I still want to see if you can get a retry case and make the same test as [#2](#).
4. The analysis of openScope(), finished() and the effects of the client-scope notifications is still needed.

**#214 - 10/15/2014 04:56 PM - Eugenie Lyzenko**

- File testcases\_evl20141015a.zip added

More investigation results.

1. There is no changes as result of setting DOWN value to 0.
2. RETRY statement does not reset dynamic frame, the screen values become not changed, dynamic widgets(only one widgets that can be used with dynamic frames) are in state was just before RETRY action(dyn\_wid\_test25.p).
3. We need to implement the finished() functionality for dynamic frames(the same as for one down static frames). Any pending actions(properly buffer closing, save unsaved data, etc) should be done on frame termination(dyn\_wid\_test26.p).
4. To be able to run actions from [#3](#) we need to apply openScope(). The good place to do this can be dynamic frame creation moment or the moment when dynamic frame is adding to the WidgetPool.

The points I need to clarify more is the effects of the client-scope notifications. To be continued tomorrow.

**#215 - 10/15/2014 05:07 PM - Greg Shah**

How do you know that in dyn\_wid\_test25.p the widget screen-value is not reset during the retry? None of the fillin widgets ever actually display formatted output on the screen so there is no visual record of what is happening. I don't see any code in the test itself to check the state of the screen-value attribute. I don't think that test is really proving anything.

Please do check in your testcases, so that we can just use bzt to access them.

**#216 - 10/16/2014 08:59 AM - Eugenie Lyzenko**

The frame scoping related tests (19-26) has been committed in bzt as 1211. I did not add dynamic widget tests there(not related to frame scoping point). I'm planning to upload all other dynamic widgets tests in uast/dynamic\_widget. Is it OK?

**#217 - 10/16/2014 01:36 PM - Eugenie Lyzenko**

I have committed in bzt the modified version of dyn\_wid\_test25.p with check for the screen values after retry. And the values are not reset. Do you think it confirm the theory we can skip reset in retry as I think?

**#218 - 10/16/2014 02:05 PM - Greg Shah**

I'm planning to upload all other dynamic widgets tests in uast/dynamic\_widget. Is it OK?

Yes, please do.

I have committed in bzt the modified version of dyn\_wid\_test25.p with check for the screen values after retry. And the values are not reset. Do you think it confirm the theory we can skip reset in retry as I think?

I think so.

**#219 - 10/16/2014 02:52 PM - Eugenie Lyzenko**

I'm planning to upload all other dynamic widgets tests in uast/dynamic\_widget. Is it OK?

Yes, please do.

Committed in bzt as 1213.

**#220 - 10/16/2014 05:47 PM - Eugenie Lyzenko**

Some consideration for handling LogicalTerminal.signalFrameScope() signals in server and client sides.

The server side schema:

1. Scope is opening - GenericFrame.openScope() calls LogicalTerminal.signalFrameScope() for MarkEntry.OPEN\_SCOPE event.

2. Scope is closing - GenericFrame.finished() calls LogicalTerminal.signalFrameScope() for MarkEntry.CLOSE\_SCOPE event.
3. Frame is need to be cleaned - GenericFrame.frameCleanup() calls LogicalTerminal.signalFrameScope() for MarkEntry.DELETED\_SCOPE event.
4. Frame is viewed - LogicalTerminal.signalFrameScope() causes the events to be added via LogicalTerminal.addMarkEntry(): MarkEntry.CAN\_HIDE\_OTHERS\_EXT for top level transaction block or MarkEntry.CAN\_HIDE\_OTHERS otherwise as the call of LogicalTerminal\$ScopeProcessor.frameViewed().

Client side processing:

The call ThinClient.markFrames(), assuming the dynamic frame is not a down frame does:

1. Clear all widgets for OPEN\_SCOPE.
2. Postpone repainting for CLOSE\_SCOPE.
3. Force deregistering of page elements for DELETED\_SCOPE.
4. Sets can hide others flag for CAN\_HIDE\_OTHERS.
5. Sets can wipe screen flag for CAN\_HIDE\_OTHERS\_EXT.

So we certainly need to handle OPEN\_SCOPE to properly prepare the frame for further usage. Not sure about CLOSE\_SCOPE action because it is not clear if the dynamic frame can leave it's scope otherwise than termination case(it is in scope while widget pool is alive, correct?). I think we do not need page frame deregisters, or we can safely call this because it there are no such element - nothing will happen. Not clear for now what purpose of the hide/wipe flags for dynamic frames(point 3, 4 above).

In general, looks like dynamic widgets and frames are designed for some special purpose like creating visual building tools, UI oriented debuggers or like this. Anyway we need testcases to uncover client side behavior, continue working tomorrow.

#### #221 - 10/17/2014 06:23 PM - Eugenie Lyzenko

4 more testcases have been committed in bsr as 1214.

Testcase dyn\_wid\_test27.p demonstrates the client side reaction for leaving the scope. The named widget pool in internal procedure is used. When the control is out of scope - the frame disappears from screen. There is no manual deletion of the widget pool or frame itself. So looks like we need to handle CLOSE\_SCOPE.

DELETED\_SCOPE with manual deleting the widget pool (dyn\_wid\_test28.p) has the same effect as CLOSE\_SCOPE handling. So we need to perform this cleanup in the first message we encounter, either DELETED\_SCOPE or CLOSE\_SCOPE.

The testcases dyn\_wid\_test29.p and dyn\_wid\_test30.p demonstrate different show/hide behavior for dynamic frames from named and default widget pools. For default case the frame is hiding just after hide statement and showing after view. The named pool frames does not react on hide statement. Until full screen repaint is calling. On the other hand if some static frame overlap the inactive dynamic frame region - dynamic frame is hiding. But may be this is a kind of Progress bug.

Looks like we have info to start implementation. More testcases might be required further to clear details but I guess we have general picture.

I think we are almost there.

I agree that dynamic widgets appear to be managed based on the widget-pool. This is consistent with the 4GL documentation. In particular, we know that when the widget-pool goes out of scope or is explicitly deleted, that the resources in the widget-pool will be deleted automatically (as if we did a DELETE WIDGET hndl. for each of the dynamic widgets in that pool.

I think this largely explains the behavior you are seeing.

My concerns are these:

1. In order to use a dynamic frame, what portions of the openScope() need to still be done? I'm sure that most of the openScope() is NOT needed. But I wonder if some of it is needed:

```
public void openScope()
{
    // push down any pending screen definitions
    LogicalTerminal.processDeferredPush();           <----- possibly needed

    // register for iterate, retry and finished notifications
    TransactionManager.registerFinalizable(this, false);   <----- NOT needed

    // associate this frame with a scope in the LT (this is safe to do
    // multiple times and must be safe because of multiple nested calls
    // to this method)

    if (scopeNesting == 0)
    {
        LogicalTerminal.registerFrameInScope(this);       <----- NOT needed

        // reset frame to its initial state
        resetFrameBuf();                                   <----- NOT needed
        pendingUpDown = 0;                                  <----- NOT needed
        hadView       = false;                             <----- NOT needed
        canHide       = true;                              <----- NOT needed
        hadCondition  = false;                             <----- NOT needed

        // the screen buffer cache needs also to be cleared
        screenBufferCache.clear();                         <----- NOT needed
    }

    LogicalTerminal.signalFrameScope(this, MarkEntry.OPEN_SCOPE);   <----- probably needed

    scopeNesting ++;                                       <----- NOT needed
}
```

I assume that we DON'T want to call openScope() for dynamic frames, since so much of that logic is not needed. But the portions that are needed should be processed somewhere. I suspect this should be at the end of createDynamicFrame().

2. Which portions of client side OPEN\_SCOPE code are needed? Is it as simple as implementing a full OPEN\_SCOPE notification?

Have you review the use of Frame.scopeNesting and the other state variables maintained in the LogicalTerminal.markFrames() method for case MarkEntry.OPEN\_SCOPE:? Note that DOWN = 0 will cause a dynamic frame to report true for isDown().

3. Which portions of client side CLOSE\_SCOPE code are needed? Is it as simple as implementing a full CLOSE\_SCOPE notification?

I expect that this should be processed when a dynamic widget is deleted. I'm pretty sure most or all of the CLOSE\_SCOPE notification is needed.

In addition, are there any other portions of finished() that are needed?

And where do we cause this notification to occur? Static frames call TransactionManager.registerFinalizable() and get a finished() call as a result, but we won't get this.

4. About DELETE\_SCOPE, you noted this previously:

I think we do not need page frame deregisters, or we can safely call this because it there are no such element - nothing will happen. Not clear for now what purpose of the hide/wipe flags for dynamic frames(point 3, 4 above).

It seems like a dynamic widget can be made into a header or footer like this:

```
CREATE FRAME hndl
  ASSIGN PAGE-TOP = true. /* or PAGE-BOTTOM = true. */
```

However, I don't think we need DELETE\_SCOPE as long as we do call DESTROY (since DESTROY calls destroyFrame() which also does call forceDeregisterPageElement()).

I think this is already handled by FrameWidget.resourceDelete() -> GenericFrame.frameCleanup() -> GenericFrame.destroy() -> LogicalTerminal.destroyFrame() -> ThinClient.destroyFrame()/MarkEntry.DESTROY.

One thing I don't understand is why GenericFrame.frameCleanup() needs to do the MarkEntry.DELETED\_SCOPE since it will . It seems like it just causes the forceDeregisterPageElement() to be processed slightly earlier than would happen in the MarkEntry.DESTROY case.

All of this should be gotten "automatically" by either an explicit DELETE WIDGET hndl. or by the implicit delete processing that occurs from the WidgetPool.addResource() that is called in DynamicWidgetFactory.create().

Constantin: do you have any thoughts to add?

## #223 - 10/20/2014 05:04 PM - Eugenie Lyzenko

2. Which portions of client side OPEN\_SCOPE code are needed? Is it as simple as implementing a full OPEN\_SCOPE notification?

Have you review the use of Frame.scopeNesting and the other state variables maintained in the LogicalTerminal.markFrames() method for case MarkEntry.OPEN\_SCOPE:? Note that DOWN = 0 will cause a dynamic frame to report true for isDown().

I think we need only frame.setInScope(true) on the client side. Because for dynamic frames there is no scope nesting, correct? We just need to mark the frame in scope on the client side. I would suggest we need the frame.resetAll() to be executed but when the frame is creating it has nothing inside so we really have nothing to reset.

As to down flag collision I suggest not consider DOWN = 0 case, for example always return false from dynamic frame isDown() call. There is no dynamic down frames.

3. Which portions of client side CLOSE\_SCOPE code are needed? Is it as simple as implementing a full CLOSE\_SCOPE notification?

I expect that this should be processed when a dynamic widget is deleted. I'm pretty sure most or all of the CLOSE\_SCOPE notification is needed.

Yes, I guess so. Except the change for Frame.scopeNesting class member itself(it is always 0 I think). I guess every change for members in Frame class is useless unless we will handle another message for this frame(DELETED\_SCOPE for example) - just because leaving the scope the dynamic frame becomes unusable.

**#224 - 10/20/2014 05:59 PM - Eugenie Lyzenko**

And where do we cause this notification to occur? Static frames call `TransactionManager.registerFinalizable()` and get a `finished()` call as a result, but we won't get this.

Combination of the `HandleOps.delete()` and frame hiding(if we can not do such cleanup just in `HandleOps.delete()`). Say we can mark the frame as pending to be closed and perform the real work when it becomes hidden.

I think this is already handled by `FrameWidget.resourceDelete()` -> `GenericFrame.frameCleanup()` -> `GenericFrame.destroy()` -> `LogicalTerminal.destroyFrame()` -> `ThinClient.destroyFrame()/MarkEntry.DESTROY`.

One thing I don't understand is why `GenericFrame.frameCleanup()` needs to do the `MarkEntry.DELETED_SCOPE` since it will. It seems like it just causes the `forceDeregisterPageElement()` to be processed slightly earlier than would happen in the `MarkEntry.DESTROY` case.

Constantin, I think you can explain the goal of this modification in `GenericFrame`: \*\* 259 CA 20131013 ... Any comments?

**#225 - 10/21/2014 07:11 AM - Eugenie Lyzenko**

All of this should be gotten "automatically" by either an explicit `DELETE WIDGET` hndl.

OK.

or by the implicit delete processing that occurs from the `WidgetPool.addResource()` that is called in `DynamicWidgetFactory.create()`.

Please clarify this. I see in this method only adding new widget to the set. Do you mean the case when we replace old widget with the new one for same handle? Or the code just should be written in `WidgetPool.addResource()` like existed in the `WidgetPool.scopeFinished()` method?

I think we need only `frame.setInScope(true)` on the client side. Because for dynamic frames there is no scope nesting, correct? We just need to mark the frame in scope on the client side. I would suggest we need the `frame.resetAll()` to be executed but when the frame is creating it has nothing inside so we really have nothing to reset.

OK.

Although there is no scope nesting, I don't know that we need to exclude it.

Please make sure it does call `resetAll()` because in the future if we add some important initialization logic there, it may be important to have that called for all frames.

Unless there is something that is important to exclude, perhaps we just execute all of the `MarkEntry.OPEN_SCOPE`.

As to down flag collision I suggest not consider `DOWN = 0` case, for example always return false from dynamic frame `isDown()` call. There is no dynamic down frames.

I'm not sure about this. Remember Constantin's finding:

if a dynamic frame is created dynamically, setting its `DOWN` to 0 will automatically compute the down relative to the window's height, when the frame is realized

The dynamic widgets don't get duplicated AND as far as we have found, you cannot put static widgets into a dynamic frame. However, the frame itself probably acts like a down frame that has no iterations. Anyway, we probably don't need this right now since it is a very unusual case. I don't want to spend time to explore what could be a very complicated area if it is not needed immediately (possibly ever). Just leave the down processing as is.

Yes, I guess so. Except the change for `Frame.scopeNesting` class member itself(it is always 0 I think). I guess every change for members in `Frame` class is useless unless we will handle another message for this frame(`DELETED_SCOPE` for example) - just because leaving the scope the dynamic frame becomes unusable.

Again, unless we really need to exclude something, let's execute all of `MarkEntry.CLOSE_SCOPE`. Please note that we are not really "leaving the frame's scope" since dynamic frames don't have a scope. They have a lifetime that is defined by the first to occur of these two events:

1. There is an explicit `DELETE_WIDGET` called for that dynamic frame.
2. The widget-pool to which the frame has been added (at creation time) goes out of its scope or is explicitly deleted. At that point, every resource in the widget-pool gets the equivalent of a `DELETE_WIDGET` implicitly called on it.

My point is that the `MarkEntry.CLOSE_SCOPE` and `MarkEntry.DESTROY` probably need to occur at the same time for dynamic frames. Perhaps we can add the `MarkEntry.CLOSE_SCOPE` to `GenericFrame.destroy()` since there is no `finished()` call for dynamic frames.

Combination of the `HandleOps.delete()` and

`GenericFrame.destroy()` gets called when `HandleOps.delete()` is executed, right. I think `GenericFrame.destroy()` is the right place.

frame hiding(if we can not do such cleanup just in `HandleOps.delete()`). Say we can mark the frame as pending to be closed and perform the real work when it becomes hidden.

Is there some special behavior we need to implement for dynamic frame hiding?

or by the implicit delete processing that occurs from the `WidgetPool.addResource()` that is called in `DynamicWidgetFactory.create()`.

Please clarify this.

Hopefully my explanation above helps. Please also read the WIDGET-POOL entry in the 4GL docs and also read the javadoc in the WidgetPool class.

I see in this method only adding new widget to the set.

The widget pool is a scoped resource. When it is explicitly deleted OR when it goes out of scope, then all resources inside it are deleted automatically. This means that the programmer does not have to write any cleanup code in the 4GL but the widgets will still be implicitly deleted.

Do you mean the case when we replace old widget with the new one for same handle?

No.

Or the code just should be written in WidgetPool.addResource() like existed in the WidgetPool.scopeFinished() method?

No. I hope the above clarifications helped. If not, ask more questions.

#### **#227 - 10/21/2014 12:32 PM - Constantin Asofiei**

Eugenie Lyzenko wrote:

And where do we cause this notification to occur? Static frames call TransactionManager.registerFinalizable() and get a finished() call as a result, but we won't get this.

Combination of the HandleOps.delete() and frame hiding(if we can not do such cleanup just in HandleOps.delete()). Say we can mark the frame as pending to be closed and perform the real work when it becomes hidden.

I think this is already handled by FrameWidget.resourceDelete() -> GenericFrame.frameCleanup() -> GenericFrame.destroy() -> LogicalTerminal.destroyFrame() -> ThinClient.destroyFrame()/MarkEntry.DESTROY.

One thing I don't understand is why GenericFrame.frameCleanup() needs to do the MarkEntry.DELETED\_SCOPE since it will . It seems

like it just causes the `forceDeregisterPageElement()` to be processed slightly earlier than would happen in the `MarkEntry.DESTROY` case.

I think at this point `DESTROY` and `DELETED_SCOPE` all called in the same way, and should be merged. For static frames accessed outside their instantiating-procedure, their default scope will survive until the frame is deleted/destroyed.

Constantin, I think you can explain the goal of this modification in `GenericFrame`: \*\* 259 CA 20131013 ... Any comments?

The goal of the H259 changes is described here: [#2196-40](#) . Mainly, is related to the fact that a static frame can be accessed outside of the instantiating-procedure, when that procedure is ran persistent. And in this case, there is no longer a matter of opening/closing scopes... I think once you obtain a handle for a widget, no more intermediate scope processing is done, when the handle for that widget is used. For static frames, the default scope will end when the frame is destroyed; for dynamic frames, a scope is opened when the frame is created and will be destroyed/closed when the frame is deleted.

#### #228 - 10/24/2014 03:29 PM - Eugenie Lyzenko

- File `evl_upd20141024a.zip` added

This is the first drop for your review of the next step in dynamic widget implementation.

Unfortunately I have found one bug that is common to static and dynamic frames. If static or dynamic frame contains nothing - it will not be displayed on the client side. For example:

```
...
def frame stfr with row 2 column 1 size 50 by 5.
view frame stfr.
...
```

While 4GL in this case shows the empty frame. And while we have this issue - we can not check if the dynamic frame works OK. But we can still work with dynamic widgets inside the static frame(if there is at least one static widget in a frame).

#### #229 - 10/24/2014 03:31 PM - Constantin Asofiei

Eugenie Lyzenko wrote:

Unfortunately I have found one bug that is common to static and dynamic frames. If static or dynamic frame contains nothing - it will not be

displayed on the client side. For example:

Marius, do you think this will be fixed with your current work?

**#230 - 10/24/2014 05:03 PM - Greg Shah**

Code Review 1024a

The code looks good. Although this should have very little risk, please still put it through runtime regression testing.

Marius had to put in a fix to get empty frames working for his GUI frame implementation in [#2416](#). That fix is what Constantin is referencing and it may be the solution to the bug you have found.

**#231 - 10/26/2014 03:05 AM - Eugenie Lyzenko**

The main part completed without regressions. Waiting for CTRL-C one to be done.

**#232 - 10/26/2014 02:50 PM - Eugenie Lyzenko**

The CTRL-C part has been completed without regressions, had to start 3-way tests separately to have positive results for them. The results for review has been copied to the shared dir(10635\_54e6e9b\_20141026\_evl.zip).

So I guess the update 1024a is ready to be checked into bzt and distribute.

**#233 - 10/27/2014 07:50 AM - Greg Shah**

Go ahead and check in/distribute.

**#234 - 10/27/2014 09:51 AM - Eugenie Lyzenko**

Committed in bzt as 10636. Will be distributed soon.

**#235 - 07/23/2015 03:39 PM - Stanislav Lomany**

For CREATE BROWSE created task branch 1791a from P2J trunk revision 10898.

**#236 - 07/23/2015 03:50 PM - Greg Shah**

Please change the branch name to 2026a. That way we can match up your specific changes with the child task.

**#237 - 07/24/2015 06:39 AM - Stanislav Lomany**

For CREATE BROWSE created task branch 2026a from P2J trunk revision 10898.

**#238 - 07/24/2015 08:13 AM - Greg Shah**

In methods\_attributes.rules, please add the new attributes (down and query) to the load\_descriptors function instead of to the main body of the file. Over time, we want the entire file to be encoded in the "new" way that is much simpler to read and to maintain.

**#239 - 07/24/2015 08:19 AM - Constantin Asofiei**

Stanislav, if you are adding support for DOWN, please make sure that FRAME:DOWN is supported too.

**#240 - 07/28/2015 01:26 PM - Stanislav Lomany**

Rebased task branch 2026a from P2J trunk revision 10901.

**#241 - 07/30/2015 06:22 AM - Stanislav Lomany**

Rebased task branch 2026a from P2J trunk revision 10902.

**#242 - 07/30/2015 08:58 AM - Greg Shah**

Did you expect to remove the query attribute from the methods\_attributes.rules in revision 10904 of branch 2026a?

Also, don't forget to update the superinterface HandleCommon.java with the new attribute interfaces.

**#243 - 07/30/2015 10:41 AM - Stanislav Lomany**

Did you expect to remove the query attribute from the methods\_attributes.rules in revision 10904 of branch 2026a?

Yes. It was already defined in the list of methods above:

```
prog.kw_query, "QueryAssociable", "getQueryAsHandle", "setQueryAsHandle"
```

**#244 - 07/30/2015 01:29 PM - Stanislav Lomany**

Is CREATE ... TRIGGERS supposed to be converted properly? Is somebody working on that? Currently

```
def var wh-tmp as widget-handle.
```

```
create button wh-tmp
  assign
    visible = true
    row = 2
  triggers:
    on choose
      message "Choosed".
  end.
```

is converted to

```
DynamicWidgetFactory.createButton(whTmp, "choose", TriggerBlock0.class, Test.this); //WRONG
```

```
public class TriggerBlock0
extends Trigger
{
  public void body()
  {
    message("Choosed");
  }
}
```

**#245 - 07/30/2015 01:32 PM - Greg Shah**

I think this should have been done previously, but it was missed.

We do need TRIGGERS phrase support to be implemented properly. It is used 52 times in 20 files. I have found usage for BUTTON and BROWSE. I did not check every usage location, so other widgets may also be affected. It seems to me that the same code should work for all forms of the statement.

Please do include this in your work.

**#246 - 08/04/2015 03:40 PM - Stanislav Lomany**

There is file `embedded_attribute_assign_rewrite.rules` which rewrites `CREATE...ASSIGN`. I've added rewrite for `CREATE...TRIGGERS` to it. I think the name of the file doesn't accurately reflect file contents after that. Should I rename the file (say, to `create_widget_rewrite.rules`) or extract new code to a new file or file name is OK?

**#247 - 08/05/2015 07:57 AM - Stanislav Lomany**

Rebased task branch 2026a from P2J trunk revision 10909.

**#248 - 08/05/2015 07:58 AM - Stanislav Lomany**

Please review task branch 2026a revision 10913.

**#249 - 08/05/2015 08:58 AM - Greg Shah**

Code Review Task Branch 2026a Revision 10913

I'm OK with the changes.

Should I rename the file (say, to `create_widget_rewrite.rules`) or extract new code to a new file or file name is OK?

The `TRIGGER_PHRASE` is also possible in the 4GL as part of variable definitions. We will have to add support for that later. I think it is best to extract the new code into a `triggers_phrase_rewrite.rules` file so that future (define variable) additions will make sense.

Eric: please review the query changes.

**#250 - 08/05/2015 11:54 AM - Stanislav Lomany**

Committed `triggers_phrase_rewrite.rules`.  
Am I right that I shouldn't run runtime regression testing?

**#251 - 08/05/2015 12:41 PM - Greg Shah**

Code Review Task Branch 2026a Revision 10914

The changes look good.

Am I right that I shouldn't run runtime regression testing?

Correct, just do conversion testing.

**#252 - 08/05/2015 01:51 PM - Stanislav Lomany**

Rebased task branch 2026a from P2J trunk revision 10910.

Conversion regression passed.

I guess, waiting for Eric to review changes in QueryAssociable?

**#253 - 08/05/2015 02:09 PM - Greg Shah**

As soon as he approves it, you can merge to trunk. He will have it done shortly.

**#254 - 08/05/2015 02:28 PM - Eric Faulhaber**

Code review 2026a/10914:

What is the purpose of the new variant setQueryAsHandle(QueryWrapper), which is defined in QueryAssociable and implemented in BrowseWidget? Is this just to allow code emitted into business logic to skip the handle instantiation when associating a query with a browse? I don't have a problem with it if that's the point, just making sure I understand correctly.

**#255 - 08/05/2015 02:37 PM - Stanislav Lomany**

Yes, that is the point.

**#256 - 08/05/2015 02:46 PM - Stanislav Lomany**

Should I distribute the update?

**#257 - 08/05/2015 02:56 PM - Eric Faulhaber**

I looked a bit deeper, and something concerns me. From BrowseWidget:

```
/**
 * Set the query associated with the given resource.
 *
 * @param   qry
 *          The query to associate.
 */
@Override
public void setQueryAsHandle(handle qry)
{
    if (qry == null || qry.isUnknown() || !qry._isValid())
    {
        UnimplementedFeature.todo("QUERY attribute setter unknown behavior not investigated.");
        return;
    }

    Object resource = qry.get();

    if (!(resource instanceof QueryWrapper))
    {
        UnimplementedFeature.todo("QUERY attribute setter wrong resource behavior not investigated.");
        return;
    }
}
```

```

        setQuery((QueryWrapper) resource);
    }

    /**
     * Set the query associated with the given resource.
     *
     * @param qry
     *      The query to associate.
     */
    @Override
    public void setQueryAsHandle(QueryWrapper qry)
    {
        setQueryAsHandle(new handle(qry));
    }

```

This idiom of wrapping a known resource in a handle which is thrown away immediately is used in several places in P2J, but it is expensive, so we should be careful with it. handle construction involves a trip through `ErrorHandler.isPendingError`, which involves 2 context-local dereferences. Each of these is expensive -- the code path goes through a lot of code in the security manager to check security context and ends with a `ThreadLocal` lookup. I see `ContextLocal.get()` show up as a bottleneck in performance profiling a lot, so I'm very sensitive to adding new code paths with this dependency if it's not absolutely necessary. Is this `ErrorHandler` service really needed in this case, or can we just call `QueryWrapper.valid()` and `BrowseWidget.setQuery(QueryWrapper)` directly? Then, `BrowseWidget.setQueryAsHandle(handle)` would invoke `setHandleAsHandle(QueryWrapper)`, instead of the other way around.

#### #258 - 08/05/2015 02:59 PM - Eric Faulhaber

Having written that, I don't know what the converted code that uses this will look like. I don't expect that this code will be hit in tight loops, so it may be just fine. Is it likely this will be called more often than just once, before a browse is used?

#### #259 - 08/05/2015 03:14 PM - Stanislav Lomany

It is not expected to be used in loops. Should I change the code to

```

public void setQueryAsHandle(QueryWrapper qry)
{
    setQuery(qry);
}

```

and distribute?

**#260 - 08/05/2015 03:23 PM - Eric Faulhaber**

Stanislav Lomany wrote:

It is not expected to be used in loops. Should I change the code to

[...]

and distribute?

Since it's not expected to be called in loops, you can leave it as is and merge. Thanks.

**#261 - 08/05/2015 03:44 PM - Stanislav Lomany**

2026a was merged to trunk rev 10911.

**#262 - 08/06/2015 12:14 PM - Greg Shah**

Now that you have had 2-3 days to review the runtime support needed for CREATE BROWSE, please provide an estimate of when you believe that can be ready for testing.

**#263 - 08/06/2015 12:55 PM - Stanislav Lomany**

Here is the set of problems so far:

1. Error if VISIBLE/HIDDEN is set prior to SENSITIVE. Doesn't affect the customer's app. Should I work on it?
2. In P2J ADD-LIKE-COLUMN cannot be called for a static query until it is opened. Requires conversion changes. Doesn't affect the customer's app. Should I work on it?
3. QUERY-PREPARE doesn't work. Is it something with my configuration or do you see the issue too?

Testcase:

```
def temp-table tt field f1 as integer.  
create tt. tt.f1 = 1.  
create tt. tt.f1 = 2.  
create tt. tt.f1 = 3.  
  
def var qh1 as handle.  
def var qh2 as handle.  
def var bh1 as handle.  
bh1 = buffer tt:handle.  
  
create query qh1.  
qh1:set-buffers(bh1).  
qh1:query-prepare("for each tt").  
qh1:query-open.  
message "place1: " + string(qh1:is-open).  
  
qh1:query-prepare("for each tt where tt.f1 > 10").  
message "place2: " + string(qh1:is-open).
```

Stacktrace:

Caused by: java.lang.NullPointerException

```
at com.goldencode.p2j.cfg.Configuration.getSchemaConfig(Configuration.java:228)
at com.goldencode.p2j.schema.SchemaLoader.<init>(SchemaLoader.java:224)
at com.goldencode.p2j.schema.SchemaDictionary.<init>(SchemaDictionary.java:725)
at com.goldencode.p2j.schema.SchemaDictionary$1.initialValue(SchemaDictionary.java:584)
at com.goldencode.p2j.schema.SchemaDictionary$1.initialValue(SchemaDictionary.java:579)
at com.goldencode.p2j.security.ContextLocal.get(ContextLocal.java:407)
at com.goldencode.p2j.security.ContextLocal.get(ContextLocal.java:367)
at com.goldencode.p2j.schema.SchemaDictionary.localInstance(SchemaDictionary.java:678)
at com.goldencode.p2j.persist.DynamicQueryHelper.generateJavaTree(DynamicQueryHelper.java:303)
at com.goldencode.p2j.persist.DynamicQueryHelper.parse(DynamicQueryHelper.java:198)
at com.goldencode.p2j.persist.DynamicQueryHelper.parseQuery(DynamicQueryHelper.java:601)
at com.goldencode.p2j.persist.QueryWrapper.prepare(QueryWrapper.java:3366)
at com.goldencode.p2j.persist.QueryWrapper.prepare(QueryWrapper.java:3320)
at com.goldencode.testcases.QueryScopeReopen$1.body(QueryScopeReopen.java:43)
at com.goldencode.p2j.util.BlockManager.processBody(BlockManager.java:7071)
at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:6893)
at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:233)
at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:215)
at com.goldencode.testcases.QueryScopeReopen.execute(QueryScopeReopen.java:20)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
```

**#264 - 08/06/2015 01:02 PM - Eric Faulhaber**

Stanislav Lomany wrote:

3. QUERY-PREPARE doesn't work. Is it something with my configuration or do you see the issue too?

It must be configuration. QUERY-PREPARE works and is very heavily tested. I don't recognize this specific error. Please debug into Configuration:228 to see what is going wrong.

**#265 - 08/06/2015 01:17 PM - Greg Shah**

Please resolve all 3 issues. When do you think you have the changes ready?

Other than those 3 problems, there is no other known work left to do before closing [#2026](#)?

**#266 - 08/06/2015 03:33 PM - Stanislav Lomany**

I found also:

4. Browse does not receive focus when created in a trigger.
5. Error on WIDGET DELETE.
6. Depending on assignment order, there may be various 4GL error messages like "Unable to set ROW-MARKERS because the BROWSE widget has been realised. (4053)". Do I need to implement them?

So far it is the complete list. I think that will take ~4 days overall.

**#267 - 08/06/2015 04:00 PM - Greg Shah**

Depending on assignment order, there may be various 4GL error messages like "Unable to set ROW-MARKERS because the BROWSE widget has been realised. (4053)". Do I need to implement them?

Yes, please do.

So far it is the complete list. I think that will take ~4 days overall.

OK, good.

**#268 - 08/07/2015 07:41 AM - Stanislav Lomany**

2. In P2J ADD-LIKE-COLUMN cannot be called for a static query until it is opened. Requires conversion changes

Eric, I suggest to emit `addBuffer(..., true)` at `DEFINE QUERY` node's place. No runtime changes are required. Example:

```
def temp-table tt field t1 as integer.
def temp-table tt2 field t2 as integer.

def query q1 for tt, tt2.

message string(query q1:num-buffers).
message string(query q1:get-buffer-handle(1):name) + " " + string(query q1:get-buffer-handle(2):name).

open query q1 for each tt, each tt2.
close query q1.
```

```
message string(query q1:num-buffers).
message string(query q1:get-buffer-handle(1):name) + " " + string(query q1:get-buffer-handle(2):name).
```

#### Conversion:

```
externalProcedure(new Block()
{
    public void body()
    {
        RecordBuffer.openScope(tt2, tt);
        query0.addBuffer(tt, true);    <-----
        query0.addBuffer(tt, true);    <-----
        message(valueOf(query0.numBuffers()));
        message(concat(valueOf(query0.bufferHandle(1).unwrap().name()), " ", valueOf(query0.bufferHandle(2)
).unwrap().name())));
        query0.assign(new CompoundQuery(true, false));
        query0.addComponent(new AdaptiveQuery(tt, (String) null, null, "tt.id asc"));
        query0.addComponent(new AdaptiveQuery(tt2, (String) null, null, "tt2.id asc"));
        query0.open();
        query0.close();
        message(valueOf(query0.numBuffers()));
        message(concat(valueOf(query0.bufferHandle(1).unwrap().name()), " ", valueOf(query0.bufferHandle(2)
).unwrap().name())));
    }
});
```

**#269 - 08/07/2015 08:58 AM - Stanislav Lomany**

Plus issue:

7. Browse with dynamic query hangs on the last record.

**#270 - 08/07/2015 09:13 AM - Stanislav Lomany**

Created task branch 2026b from P2J trunk revision 10916.

**#271 - 08/07/2015 05:11 PM - Stanislav Lomany**

Eric, I found no explicit way to set a dynamic query SCROLLING/NON-SCROLLING. As testing shows, it is scrolling. I've updated P2J accordingly.

**#272 - 08/07/2015 05:22 PM - Eric Faulhaber**

Stanislav Lomany wrote:

Eric, I suggest to emit `addBuffer(..., true)` at `DEFINE QUERY` node's place.

This will double up the buffers in the `QueryWrapper.buffers` list. What problem does this solve?

**#273 - 08/07/2015 05:30 PM - Stanislav Lomany**

It allows `NUM-BUFFERS` and `BUFFER-HANDLE` to work before the query was opened.

**#274 - 08/07/2015 05:53 PM - Eric Faulhaber**

Unless you change the behavior of `QueryWrapper.addComponent`, though, it will break their use (and possibly other APIs) after the `addComponent` calls. I don't think this is just a conversion change, but runtime as well. Please investigate the runtime impact before introducing this. Note that in many static query cases, we don't have a `DEFINE QUERY` statement at all, just `OPEN QUERY`. So, `addComponent` still has to work on its own, without assuming a previous `addBuffer` call.

**#275 - 08/10/2015 07:13 AM - Stanislav Lomany**

I'm making changes only to static query case:

1. `OPEN QUERY` without `DEFINE QUERY` - no conversion changes.
2. `OPEN QUERY` with `DEFINE QUERY` - my aim is to add buffers for the period after `DEFINE` and before `OPEN`. I'll add `addBuffer` calls on `DEFINE`, after that `query.assign` on `OPEN` resets buffers before potential `addComponent` calls.

**#276 - 08/10/2015 01:48 PM - Stanislav Lomany**

There is a problem related to `DynamicWidgetFactory.pushScreen(Un)lock`.

1. Conversion assumes that

```
CREATE x.  
ASSIGN x.y = ....
```

has different behavior in comparison with

```
CREATE x ASSIGN x.y = ....
```

2. The problem I need to fix is related to the second case: `SENSITIVE` attribute can be set to `TRUE` in one batch (using `DynamicWidgetFactory.pushScreen(Un)lock`) with `FRAME` handle. When screen definitions are pushed, the new widget is added, and `WidgetRegistry.addWidgetsToFrame` disables it by default discarding `SENSITIVE` value:

```
// make widget disabled by default  
  
if (nextWidget.isEnabled())  
    nextWidget.setEnabled(false);
```

I'm not sure how to fix this problem. Maybe set correct enabled value for dynamic widgets?

**#277 - 08/10/2015 01:55 PM - Constantin Asofiei**

Stanislav Lomany wrote:

There is a problem related to `DynamicWidgetFactory.pushScreen(Un)lock`.

1. Conversion assumes that  
[...]  
has different behavior in comparison with  
[...]

This is expected, as `ASSIGN` can contain multiple frames/widgets/etc and we can't know which one to push/unlock (or even if the handle actually refers a widget). With `CREATE ... ASSIGN`, we know for sure the handle refers a widget.

1. The problem I need to fix is related to the second case: `SENSITIVE` attribute can be set to `TRUE` in one batch (using `DynamicWidgetFactory.pushScreen(Un)lock`) with `FRAME` handle. When screen definitions are pushed, the new widget is added, and `WidgetRegistry.addWidgetsToFrame` disables it by default discarding `SENSITIVE` value:  
[...]  
I'm not sure how to fix this problem. Maybe set correct enabled value for dynamic widgets?

Yes, I think for dynamic widgets the `// make widget disabled by default` code should not be executed.

**#278 - 08/10/2015 05:45 PM - Stanislav Lomany**

Plus issue:

8. Error on browse re-creation.

**#279 - 08/10/2015 09:31 PM - Eric Faulhaber**

Stanislav Lomany wrote:

I'm making changes only to static query case:

1. `OPEN QUERY` without `DEFINE QUERY` - no conversion changes.
2. `OPEN QUERY` with `DEFINE QUERY` - my aim is to add buffers for the period after `DEFINE` and before `OPEN`. I'll add `addBuffer` calls on `DEFINE`, after that `query.assign` on `OPEN` resets buffers before potential `addComponent` calls.

OK, I'm fine with this approach.

**#280 - 08/11/2015 06:33 PM - Stanislav Lomany**

Plus issue:

9. Error on ctrl-c processing.

**#281 - 08/12/2015 01:54 PM - Stanislav Lomany**

Rebased task branch 2026b from P2J trunk revision 10919.

**#282 - 08/16/2015 07:50 PM - Stanislav Lomany**

Rebased task branch 2026b from P2J trunk revision 10923.

**#283 - 08/18/2015 10:50 AM - Stanislav Lomany**

As I suggested, concept of "realized" browse (used in "Unable to set ATTRIBUTE because the BROWSE widget has been realised" error messages) differs from P2J config.[was]Realized behavior. It P2J it is set when a widget becomes visible.

For browse rules are: if the FRAME attribute is set, it is "realized" when VISIBLE is set. Otherwise it is "realized" in WAIT-FOR.

And I'm pretty sure that the complete set of rules is more quirky that I described.

Because we will not see these errors in production, I suggest to create a separate quirk-task and close 2026.

**#284 - 08/18/2015 11:59 AM - Greg Shah**

For browse rules are: if the FRAME attribute is set, it is "realized" when VISIBLE is set. Otherwise it is "realized" in WAIT-FOR.

I assume you will implement this "basic" approach.

Because we will not see these errors in production, I suggest to create a separate quirk-task and close 2026.

Any quirks outside of the above behavior can be deferred.

**#285 - 08/19/2015 03:57 AM - Constantin Asofiei**

Stanislav Lomany wrote:

As I suggested, concept of "realized" browse (used in "Unable to set ATTRIBUTE because the BROWSE widget has been realised" error messages) differs from P2J config.[was]Realized behavior. It P2J it is set when a widget becomes visible.

For browse rules are: if the FRAME attribute is set, it is "realized" when VISIBLE is set. Otherwise it is "realized" in WAIT-FOR.

And I'm pretty sure that the complete set of rules is more quirky than I described.

Because we will not see these errors in production, I suggest to create a separate quirk-task and close 2026.

There is already the [#2572](#) task for the full support for the realized flag - I think it's better to post these notes there.

#### #286 - 08/19/2015 01:04 PM - Stanislav Lomany

Constantin, thanks, I'll make reference to this task there.

OK, here is the correct set of rules:

1. Browse is "realized" when it becomes visible. That is an arguable statement, but I think at this point this is OK approach.
2. Setting a widget (static or dynamic) VISIBLE displays the parent frame. - That probably deserves a separate task.
3. You are free to change TITLE of a realized browse with a title. But you cannot set a TITLE for a realized browse without title.

#### #287 - 08/19/2015 04:38 PM - Stanislav Lomany

Guys, at this point config.realized is set in AbstractWidget.\_setVisible. Setting of config.visible attribute doesn't always go thru that path. What do you think if I'll add this to AbstractWidget.afterConfigUpdateBase?

```
public void afterConfigUpdateBase(WidgetConfig beforeUpdate)
{
    WidgetConfig c = config();

    if (c == null)
    {
        return;
    }

    setVisible(c.visible);
    if (c.visible && !c.realized) <--
    { <--
        c.realized = true; <--
        c.wasRealized = true; <--
    } <--
}
```

#### #288 - 08/20/2015 02:32 PM - Greg Shah

I don't know the answer to the question in note 287. It seems OK to me.

Constantin/Hynek: what do you think?

**#289 - 08/20/2015 02:56 PM - Hynek Cihlar**

Indeed, the management of realized flag is a bit messy.

Stanislav, your solution seems to be ok when visible config field is set on the server. For the case when it is set on the client a similar logic must execute on the server.

**#290 - 08/20/2015 03:12 PM - Stanislav Lomany**

For the case when it is set on the client a similar logic must execute on the server.

There is no need to do it explicitly, it is done automatically in pushScreenDefinitions trip.

**#291 - 08/20/2015 03:30 PM - Hynek Cihlar**

Stanislav Lomany wrote:

For the case when it is set on the client a similar logic must execute on the server.

There is no need to do it explicitly, it is done automatically in pushScreenDefinitions trip.

Interesting, where exactly does it happen?

**#292 - 08/20/2015 04:04 PM - Stanislav Lomany**

Guys, I want to have a conversation about pushScreenLock/pushScreenUnlock. AFAIK what it does is deferring pushScreenDefinitions until pushScreenUnlock so CREATE...ASSIGN is executed in a batch rather than running assignments one by one. Consider a testcase:

```
def var h as handle.  
  
def frame fr WITH SIZE 80 BY 20.  
view frame fr.  
  
function func returns character:  
    pause message "in func".  
    return "Func label".  
end.
```

```
CREATE button h assign
    label = "Default label"
    frame = frame fr:handle
    visible = true
    label = func().
```

```
pause message "end".
```

In 4GL it *first* displays the frame (handles visible = true) and then runs func. Unfortunately in P2J it causes NPE:

```
Caused by: java.lang.NullPointerException
    at com.goldencode.p2j.ui.ButtonWidget.labelSizeCheck(ButtonWidget.java:1502)
    at com.goldencode.p2j.ui.GenericWidget.setLabel(GenericWidget.java:1052)
    at com.goldencode.p2j.ui.GenericWidget.setLabel(GenericWidget.java:1007)
```

so I made an equivalent testcase with a BROWSE which works fine only without pushScreenLock/pushScreenUnlock:

```
def temp-table tt field f1 as integer.
def query q for tt scrolling.
open query q for each tt.
def var h4 as handle.
```

```
def button btn.
def frame fr4 btn WITH SIZE 80 BY 20.
view frame fr4.
```

```
function func returns character:
    pause message "in func".
    return "Func title".
end.
```

```
CREATE BROWSE h4 assign
    width = 40
    height = 10
    query = query q:handle
    title = "Some title"
    frame = frame fr4:handle
    visible = true
    title = func()
    .
```

```
pause message "end".
```

Any ideas?

**#293 - 08/20/2015 04:21 PM - Stanislav Lomany**

There is no need to do it explicitly, it is done automatically in pushScreenDefinitions trip.

Interesting, where exactly does it happen?

AspectJ handles it.

**#294 - 08/20/2015 04:34 PM - Hynek Cihlar**

Stanislav Lomany wrote:

There is no need to do it explicitly, it is done automatically in pushScreenDefinitions trip.

Interesting, where exactly does it happen?

AspectJ handles it.

You probably mean ConfigFieldSetterAspect. This aspect registers config changes on server or client and helps to synchronize the changes with client or server. However it certainly doesn't contain business logic like 'when config field A is set with a certain value make sure the config field B is set with a certain value' or translated to your case 'when config field visible is set on the client make sure config field realized is set on the server'.

Anyway if your case doesn't require synchronizing realized on the server after visible was set on the client, then I wouldn't bother to put the synchronization logic on the server.

#### #295 - 08/20/2015 04:40 PM - Constantin Asofiei

Stanislav Lomany wrote:

Guys, I want to have a conversation about pushScreenLock/pushScreenUnlock. AFAIK what it does is deferring pushScreenDefinitions until pushScreenUnlock so CREATE...ASSIGN is executed in a batch rather than running assignments one by one. Consider a testcase:

[...]

In 4GL it *first* displays the frame (handles visible = true) and then runs func. Unfortunately in P2J it causes NPE:

There are two issues here:

1. LABEL attribute assignment causes NPE if it is set BEFORE the dynamic button is attached to a frame (NPE happens regardless if the LABEL assignment is part of CREATE or not, what it matters if the BUTTON is currently attached or not).
2. pushScreenLock/pushScreenUnlock should be emitted for CREATE ... ASSIGN only if the attribute assignment doesn't contain function calls - so we are sure that the handle/frame is not used (and the user can not interfere with it) until pushScreenUnlock is called.

#### #296 - 08/20/2015 04:59 PM - Constantin Asofiei

Greg/Stanislav: the usage of the GenericWidget.frame field (throughout the server-side widget sub-classes) should be checked and if the frame is used when i.e. an attribute is set determine what is the behavior if the attribute is set before the widget is attached to a frame (like in the BUTTON:LABEL example) and fix it.

#### #297 - 08/20/2015 05:23 PM - Stanislav Lomany

Constantin, can you provide a testcase that fails without pushScreenLock/Unlock?

#### #298 - 08/20/2015 05:26 PM - Constantin Asofiei

Stanislav Lomany wrote:

Constantin, can you provide a testcase that fails without pushScreenLock/Unlock?

```
def var h as handle.
```

```
def frame fr WITH SIZE 80 BY 20.  
view frame fr.
```

```
function func returns character:  
    pause message "in func".  
    return "Func label".  
end.
```

```
CREATE button h.  
h:label = "Default label". /* the NPE is here */  
h:frame = frame fr:handle.  
h:visible = true.  
h:label = func().
```

```
pause message "end".
```

**#299 - 08/20/2015 05:33 PM - Stanislav Lomany**

I mean the following:

1. A testcase that includes pushScreenLock after conversion (i.e. testcase should have CREATE..ASSIGN).
2. If we manually remove pushScreenLock from the converted code we will get a bug.

**#300 - 08/21/2015 02:40 AM - Constantin Asofiei**

Stanislav Lomany wrote:

I mean the following:

1. A testcase that includes pushScreenLock after conversion (i.e. testcase should have CREATE..ASSIGN).
2. If we manually remove pushScreenLock from the converted code we will get a bug.

The pushScreenLock/unlock are just an optimization in P2J - the 4GL code doesn't batch the assignments in CREATE ... ASSIGN. So, with or without these, P2J should behave the same.

**#301 - 08/21/2015 05:16 AM - Stanislav Lomany**

Unfortunately I think this optimization is incorrect. I will turn it off for browse because of the behavior differences it brings. It would be a good choice to turn it off for all widgets, but I guess it neutralizes P2J bugs caused by setting an attribute before a frame is assigned. So I will create a separate task for all that.

**#302 - 08/21/2015 05:52 AM - Constantin Asofiei**

Stanislav Lomany wrote:

Unfortunately I think this optimization is incorrect. I will turn it off for browse because of the behavior differences it brings. It would be a good choice to turn it off for all widgets, but I guess it neutralizes P2J bugs caused by setting an attribute before a frame is assigned. So I will create a separate task for all that.

Are you referring to this test in note 292 with this create?

```
CREATE BROWSE h4 assign
    width = 40
    height = 10
    query = query q:handle
    title = "Some title"
    frame = frame fr4:handle
    visible = true
    title = func()
.
```

If so, the only issue is the title = func() - please check without this assignment and let me know what the differences are (if any).

**#303 - 08/21/2015 05:58 AM - Stanislav Lomany**

func() is just a way to show that assignments are executed one by one. The differences I'm taking about is that after browse.frame is assigned and browse.visible is on, browse is considered to be "realized". Following assignments in CREATE..ASSIGN may display "browse is already realized" error messages.

**#304 - 08/21/2015 07:56 AM - Constantin Asofiei**

Stanislav Lomany wrote:

func() is just a way to show that assignments are executed one by one. The differences I'm taking about is that after browse.frame is assigned and browse.visible is on, browse is considered to be "realized". Following assignments in CREATE..ASSIGN may display "browse is already realized" error messages.

I would leave the push lock/unlock there and change the runtime this way: if the widget gets "realized", call GenericWidget.setScreenLock(false) before realizing the widget - this way all changes are pushed and the widget attribute assignments move from "batch" to "immediate" from that point forward and the next pushScreenUnlock will be a no-op.

**#305 - 08/21/2015 01:39 PM - Stanislav Lomany**

realized is set on the client side. Well, I can check locked && visible && !realized on the server side, but wouldn't it be addition of another level of complexity for slight optimization? I've already fixed couple of bugs in pushScreenLock mechanism.

**#306 - 08/21/2015 02:13 PM - Stanislav Lomany**

Rebased task branch 2026a from P2J trunk revision 10927.

**#307 - 08/21/2015 03:48 PM - Constantin Asofiei**

Stanislav, go ahead and disable the pushScreenLock/Unlock completely - comment out the code in embedded\_attribute\_assign\_rewrite.rules:76 and leave a comment behind for the reason why this was disabled.

**#308 - 08/22/2015 07:41 AM - Stanislav Lomany**

Please review task branch 2026b revision 10939.

**#309 - 08/23/2015 06:06 PM - Stanislav Lomany**

Rebased task branch 2026b from P2J trunk revision 10928.

**#310 - 08/24/2015 09:25 AM - Greg Shah**

Code Review Task Branch 2026b Revision 10940

This is really good.

The only thing I see here is that GenericFrame.deleteDynamicWidget() has browse-specific code, which is not optimal. I think this would be better done by adding an empty cleanup() method to GenericWidget. Then BrowseWidget would override it. The browse column deletion would be done inside BrowseWidget.cleanup().

Eric: would you please review the query-related changes?

### #311 - 08/24/2015 11:41 AM - Eric Faulhaber

Code review 2026b/10940:

- query\_associations.rules: please refresh my memory...why were we hiding the DEFINE\_QUERY node's children, and why is this no longer necessary now? I assume this has to do with the addBuffer change, but isn't there already a QueryWrapper reference on which to invoke those calls?
- P2JQuery.java: thanks for adding my missing javadoc for the optimize method.
- WeakList.java: please update the class javadoc to reflect the addition of the remove method.

### #312 - 08/24/2015 03:26 PM - Stanislav Lomany

- query\_associations.rules: please refresh my memory...why were we hiding the DEFINE\_QUERY node's children, and why is this no longer necessary now? I assume this has to do with the addBuffer change, but isn't there already a QueryWrapper reference on which to invoke those calls?

Previously we discarded all information about buffers defined in DEFINE\_QUERY (i.e. we suppressed emission of references to buffers) and now we pass these references to addBuffer.

### #313 - 08/24/2015 05:37 PM - Greg Shah

Code Review Task Branch 2026b Revision 10941

It looks good. Please get it regression tested (both conversion and runtime). If it passes without needing any modifications, then you can go ahead and merge to trunk.

### #314 - 08/25/2015 12:27 PM - Stanislav Lomany

(copy from email) There is the regression. Now I got the sense of the comment in GenericFrame.makeFrameVisible:

```
// direct access; for some reason, this code doesn't push the visibility change to the client
c.visible = true;
```

If screen definitions are pushed, as it required for realize to work properly, state of the widgets not displayed with DISPLAY becomes "initialized" and their contents are drawn while they shouldn't. E.g.

```
form i j with frame f1.
display i with frame f1.
```

will display j value too.

### #315 - 08/25/2015 01:42 PM - Stanislav Lomany

I don't like the fact that widget state is modified just by pushing definitions. So I think it is time to put into action the following comment:

```
public static void ConfigHelper.setModified(WidgetConfig cfg, boolean modified)
{
    cfg.modified = modified;

    // TODO: should this always get unilaterally set or should this only get set when
    //         modified changes from false to true?
    cfg.state = ScreenBuffer.CHANGED;
}
```

Committed 2026b revision 10942.

### #316 - 08/25/2015 03:13 PM - Greg Shah

Code Review Task Branch 2026b Revision 10942

I was hoping for more comments in makeFrameVisible() rather than less. Otherwise, I'm fine with the changes.

How is testing going?

### #317 - 08/26/2015 07:49 AM - Stanislav Lomany

In addition to the "value is displayed while it shouldn't" (1) changes in makeFrameVisible also brought border drawing issue (2) in majic. Change in setModified fixes issue (1) but brings "in redirected mode value is not displayed while it should" (3) issue. So we are dealing with a very sensitive code. Any help appreciated.

### #318 - 08/26/2015 08:48 AM - Constantin Asofiei

Stanislav, please specify some of the MAJIC tests which fail for the 3 issues in note 317 and also a 4GL test related to the 2026b branch changes you are testing.

I'll take a look and see what I find.

### #319 - 08/26/2015 10:16 AM - Stanislav Lomany

The following testcase is for

(1) - k should have no value

(2) report.txt should be "Employee: 0"

```
def var i as integer init 0.
def var k as integer init 0.
```

```

form i k with frame f1.
display i with frame f1.

def stream rpt.
output stream rpt to report.txt.

form header
    "Employee:"      at 1
    i
    with page-top width 132 no-labels no-box no-attr-space
    frame f-opt132.
view stream rpt frame f-opt132.
output stream rpt close.

```

To reveal (3) in majic, call function browser (?).

Browse testcase which I'm trying to put to work:

```

def var h as handle.

def button btn.
def frame fr btn WITH SIZE 80 BY 20 title "Realized vis before".
view frame fr.

CREATE BROWSE h assign
    x = 10
    y = 10
    width = 40
    height = 10
    title = "Browse title"
    visible = true
    frame = frame fr:handle
    .

h:row-markers = true.
pause message "place 1".
view frame fr.
h:row-markers = true.
pause message "place 2".

```

Browse should be realized and drawn at view frame fr. This is done by setting realized flag at makeFrameVisible and avoiding premature drawing in BrowseImpl.draw:

```

if (config.dynamic && !config.realized)
{
    return;
}

```

**#320 - 08/27/2015 05:16 AM - Constantin Asofiei**

Stanislav, I think you are managing the VISIBLE attribute wrong. Even if you set it in the CREATE BROWSE, the VISIBLE attribute remains "no" until the browse is displayed or the VISIBLE is set AFTER the browse is attached to the frame; setting the VISIBLE attr when the widget is not attached to a frame is a no-op. This interferes with how P2J processes VIEW - it sees the browse already marked as visible and no longer draws/realizes it.

So: I've removed the changes in ConfigHelper.setModified, GenericFrame.makeFrameEnable and makeFrameVisible, changed GenericFrame.setVisible to this:

```
public void setVisible(boolean visible)
{
    if (frame == null)
    {
        // no-op if frame is not set
        return;
    }
}
```

and things seem stable now.

**#321 - 08/27/2015 06:58 AM - Stanislav Lomany**

Constantin, many thanks! To be sure - are you talking about GenericWIDGET.setVisible?

**#322 - 08/27/2015 07:23 AM - Constantin Asofiei**

Stanislav Lomany wrote:

Constantin, many thanks! To be sure - are you talking about GenericWIDGET.setVisible?

Ah, correct, is GenericWidget, not GenericFrame.

**#323 - 08/27/2015 04:11 PM - Greg Shah**

Code Review Task Branch 2026b Revision 10944

I'm good with the changes. If they pass testing, please merge to trunk.

How is regression testing going?

**#324 - 08/27/2015 04:47 PM - Stanislav Lomany**

There is single minor regression. I've fixed it and committed the fix. Re-running regression testing.

**#325 - 08/27/2015 04:56 PM - Greg Shah**

Code Review Task Branch 2026b Revision 10945

The change is good. Merge to trunk if it passes.

**#326 - 08/28/2015 12:26 PM - Stanislav Lomany**

2026b has been committed into the trunk as bzt revision 10929.

**#327 - 08/28/2015 01:01 PM - Greg Shah**

- Status changed from New to Closed

**#328 - 09/03/2015 06:47 AM - Stanislav Lomany**

Fixed conversion regression brought by 2026b.  
Please review task branch 2026c, revision 10931.

**#329 - 09/04/2015 09:41 AM - Stanislav Lomany**

2026c has been committed into the trunk as bzt revision 10931.

**#330 - 11/16/2016 12:12 PM - Greg Shah**

- Target version changed from Milestone 12 to GUI Support for a Complex ADM2 App

**#331 - 01/03/2018 02:11 PM - Greg Shah**

- Related to Feature #3246: reduce the amount of data being sent to the client-side when an UI attribute is being changed added

**Files**

ca_upd20130222f.zip	107 KB	02/22/2013	Constantin Asotiei
ca_upd20130222i.zip	109 KB	02/22/2013	Constantin Asotiei
ca_upd20130222k.zip	109 KB	02/22/2013	Greg Shah
err.log	5.53 KB	11/19/2013	Vadim Gindin
widget_del.p.ast	8.32 KB	11/25/2013	Vadim Gindin
vig_upd20131127a.zip	53.2 KB	11/26/2013	Vadim Gindin
vig_upd20131130a.zip	354 KB	11/29/2013	Vadim Gindin
vig_upd20131204a.zip	415 KB	12/04/2013	Vadim Gindin
vig_upd20131206a.zip	415 KB	12/06/2013	Vadim Gindin
vig_upd20131217a.zip	416 KB	12/18/2013	Vadim Gindin
evl_upd20140731a.zip	2.53 KB	07/31/2014	Eugenie Lyzenko
testcases_20140804a.zip	5.69 KB	08/04/2014	Eugenie Lyzenko
evl_upd20140810a.zip	130 KB	08/10/2014	Eugenie Lyzenko
evl_upd20140811a.zip	253 KB	08/11/2014	Eugenie Lyzenko
evl_upd20140811b.zip	253 KB	08/11/2014	Eugenie Lyzenko
evl_upd20140813a.zip	289 KB	08/13/2014	Eugenie Lyzenko
evl_upd20140814a.zip	275 KB	08/14/2014	Eugenie Lyzenko
evl_upd20140814b.zip	279 KB	08/14/2014	Eugenie Lyzenko
evl_upd20140815a.zip	295 KB	08/15/2014	Eugenie Lyzenko
evl_upd20140816a.zip	295 KB	08/16/2014	Eugenie Lyzenko
evl_upd20140818a.zip	295 KB	08/18/2014	Eugenie Lyzenko
evl_upd20140819a.zip	295 KB	08/19/2014	Eugenie Lyzenko
evl_upd20140820a.zip	295 KB	08/20/2014	Eugenie Lyzenko

evl_upd20140820b.zip	295 KB	08/20/2014	Eugenie Lyzenko
evl_upd20140821a.zip	280 KB	08/22/2014	Eugenie Lyzenko
evl_upd20140827a.zip	299 KB	08/27/2014	Eugenie Lyzenko
evl_upd20140828a.zip	302 KB	08/29/2014	Eugenie Lyzenko
evl_upd20140829a.zip	311 KB	08/29/2014	Eugenie Lyzenko
evl_upd20140829b.zip	6.52 KB	08/29/2014	Eugenie Lyzenko
evl_upd20140901a.zip	311 KB	09/01/2014	Eugenie Lyzenko
evl_upd20140901b.zip	311 KB	09/01/2014	Eugenie Lyzenko
evl_upd20140904a.zip	296 KB	09/04/2014	Eugenie Lyzenko
evl_upd20140917a.zip	295 KB	09/17/2014	Eugenie Lyzenko
evl_upd20140918a.zip	310 KB	09/18/2014	Eugenie Lyzenko
evl_upd20140919a.zip	310 KB	09/19/2014	Eugenie Lyzenko
evl_upd20140927a.zip	153 KB	09/27/2014	Eugenie Lyzenko
evl_upd20140928a.zip	153 KB	09/28/2014	Eugenie Lyzenko
evl_upd20140929a.zip	162 KB	09/29/2014	Eugenie Lyzenko
evl_upd20141010a.zip	8.01 KB	10/10/2014	Eugenie Lyzenko
testcases_evl20141013a.zip	11.3 KB	10/13/2014	Eugenie Lyzenko
testcases_evl20141015a.zip	13.5 KB	10/15/2014	Eugenie Lyzenko
evl_upd20141024a.zip	214 KB	10/24/2014	Eugenie Lyzenko