Database - Feature #2692

Feature # 2619 (Closed): misc database features needed for a GUI app

table-level VALEXP/VALMSG support

09/08/2015 10:58 AM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eric Faulhaber	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	GUI Support for a Complex ADM2 App		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to User Interface - Feature #3281: implement some frame options	**Closed**	**04/27/2017**

History

#1 - 09/08/2015 11:10 AM - Greg Shah

Reported by a customer:

-------- Forwarded Message --------
Subject: Re: Progress database table validation
Date: Thu, 3 Sep 2015 17:23:41 +0100
From: Guy Mills ...
To: Eric Faulhaber <ecf@goldencode.com>
CC: ...

Hi,

I'm going to look at this tomorrow.

From tests I did with Alex, It seems that Progress magically compiles the code and message from the DB into the equivalent of the VALIDATE option of the DELETE statement. What would be nice, is when we do a compile preprocess, that Progress actually put this into the the preprocessed code (hmph).

Eric, if you're going to support this, then I'd suggest that's what you do (even though Progress doesn't).

For now, I'm going to see whether it will be possible to identify all the DELETE's on those tables, and insert the VALIDATE option. Failing that, I'll see whether we can add the logic into the existing DELETE triggers (most/all tables have them for potential auditing purposes) - returning an ERROR from a trigger also causes the update to fail (normally).

btw any VALIDATE option added explicitly to a DELETE statement overrides the DB validation - hence why you can bypass the DB validation by using DELETE <blah> VALIDATE heh heh.

Regards,
Guy

The automatic compilation done by Progress is for a table-level VALEXP in an ADD TABLE definition in a schema (.df) file. The VALMSG is also used. More details from the customer:

-------- Forwarded Message --------
Subject: Re: Progress database table validation
Date: Tue, 8 Sep 2015 15:18:53 +0100
From: Guy Mills ...
To: Eric Faulhaber <ecf@goldencode.com>
CC: ...

Hi Eric,

...

Just as a placeholder/reminder - this is the current situation (as I understand it):

	Progress	P2J
Explicit Delete	.v validation used	Allowed
Dynamic Delete	ERROR	Allowed

and this is what I guess you want to happen:

	Progress	Future P2J
Explicit Delete	.v validation used	.v validation used
Dynamic Delete	ERROR	ERROR

Like I mentioned previously, the DB validation actually maps to a hidden VALIDATE option on explicit DELETE statements - which I think would be better if you explicitly put into your preprocessed code (.cache files).

NB. I discovered that if there is no VALMSG (e.g. as we have on the some_table table), this corrolates to a message value of the unknown value (?).

e.g. DELETE some_table VALIDATE.

Regards,
Guy

#2 - 09/08/2015 11:13 AM - Greg Shah

We already support the proper preprocessing of these .v includes into the schema. The issue here is simply that they are ignored today.

Guy prefers these be made explicit in all locations where there is a DELETE for the matching table, but I my concern is that this will be a maintenance nightmare. Any change would require they all be modified. It seems best if we emit this into a form that is common code and call it explicitly from the runtime. Perhaps this code gets emitted in the DMO implementation class and is registered from there with the runtime. I don't see any reason to change the business logic.

#3 - 10/09/2015 12:50 PM - Eric Faulhaber

- Status changed from New to WIP

- Assignee set to Eric Faulhaber

Greg Shah wrote:

> Guy prefers these be made explicit in all locations where there is a DELETE for the matching table, but I my concern is that this will be a maintenance nightmare. Any change would require they all be modified.

Don't we handle all the field-level, schema-based validation expressions by emitting them into the business logic today?

#4 - 10/09/2015 01:09 PM - Greg Shah

> Don't we handle all the field-level, schema-based validation expressions by emitting them into the business logic today?

We honor schema-specified VALEXP/VALMSG expressions for UI widgets that are defined from a field OR which were defined from something LIKE a field. These expressions are indeed emitted to the business logic. They emit as an inner class and we then register them as validations with the UI processing.

This has nothing to do with database processing in any way. The expressions may or may not have database access in them. We don't care and don't know. They are just expressions that evaluate to a logical (VALEXP) or to a character (VALMSG).

#5 - 10/09/2015 02:59 PM - Eric Faulhaber

The problem with inlining validation expressions in the DMO implementation classes is that, as I understand it, the expressions can be arbitrarily complex. As a result, they may require context which the current conversion only creates for a business logic class. Consider, for instance, an expression which contains a CAN-FIND (possibly for a different table). This will require a record buffer and buffer scoping at minimum, to support the associated FindQuery. Who knows what else we'll encounter in these expressions? Right now, the DMO implementation classes are still simple (-ish) beans by design, but adding this capability adds new dependencies on the runtime from within the bean. I'm not interested in doing that. These have to be constructed and populated easily and quickly, both from within Hibernate and within P2J directly; adding any overhead/complexity to that is not good.

Maybe a way to centralize this is to emit a dedicated business logic class which uses BufferValidator internally. It could be invoked from RecordBuffer.delete, using a well-known naming convention (or DMO method) to determine which validation class to invoke.

I'd have to figure out how to deal with buffers and scoping issues. For example, if a buffer for a separate table is referenced in the expression, we could define all these in the invoking business logic class, but that's not very good encapsulation and it would complicate the interface to the validation class. If we don't want to pollute the business logic class from which the delete was invoked with these "extra" references and pass them to the validation class as parameters, we'll need to define them inside the validation class itself to get it to compile. The current buffer would be passed in as a parameter from RecordBuffer.delete. What about the block scaffolding to open/close scopes on these "extra" buffers? Can't visualize that just yet...

This seems like a good bit more work than emitting the expressions directly into existing business logic wherever a record is deleted, but it's probably a better long term approach. OTOH, it may be overkill for trivial validation expressions. Not sure yet if it's worth doing the simpler approach (emit everywhere) first and deferring the more complex approach, or biting the bullet now. My inclination is to go with the emit everywhere approach first. Studying the results of that approach may inform a better long-term design.

#6 - 10/09/2015 03:33 PM - Greg Shah

> as I understand it, the expressions can be arbitrarily complex

Can they reference context that exists in the business logic code that is executing the DELETE?

> Consider, for instance, an expression which contains a CAN-FIND (possibly for a different table). This will require a record buffer and buffer scoping at minimum, to support the associated FindQuery. Who knows what else we'll encounter in these expressions?

Does Progress implement some implicit buffers here or is it assuming that the programmer will implement the buffers and context needed in the procedure(s) that execute DELETE for that table?

> My inclination is to go with the emit everywhere approach first.

I agree, though I think you would want to implement it as an inner class and reuse it within a given file in case there are more than one DELETE for that table in that file. If the expressions can access arbitrary context from the business logic, then in fact it may be necessary to emit into the business logic.

#7 - 10/09/2015 04:34 PM - Eric Faulhaber

Greg Shah wrote:

> Can they reference context that exists in the business logic code that is executing the DELETE?

Don't know; I haven't gotten into test cases yet. If so, that could make it quite complicated to implement the centralized approach, and the initial approach might be all we can do after all.

> Does Progress implement some implicit buffers here or is it assuming that the programmer will implement the buffers and context needed in the procedure(s) that execute DELETE for that table?

The validation expression has to evaluate to a logical result. So, I can't see how you could structure a validation expression to include regular queries, FINDs, FOR loops, or anything like that. This naturally will limit the buffers to those used by CAN-FINDs. We've determined before that CAN-FIND does not create any buffer scopes in Progress (however, our implementation needs them). If you can reference context from the enclosing business

logic, that would enable calling UDFs with arbitrary content, making it much more complex.

Test cases will tell...

#8 - 10/20/2015 04:07 PM - Eric Faulhaber

Greg Shah wrote:

> Can they reference context that exists in the business logic code that is executing the DELETE?

Turns out, yes; the validation expression can reference a user-defined function implemented in the current procedure, and it can reference a buffer for a separate table, when that buffer is in scope at the time the delete statement is executed. Looks like we have to emit the validation expressions into the business logic.

#9 - 10/20/2015 04:15 PM - Greg Shah

That suggests that we must emit a different instance in each business logic file that uses the DELETE statement on that table AND where that DELETE doesn't have an overriding VALIDATE.

The easiest way to do this is to actually manufacture a VALIDATE clause on those DELETE statements in the AST during post-parse-fixups. Then let everything else emit as normal.

#10 - 10/21/2015 05:23 PM - Eric Faulhaber

Our current strategy of parsing validation expressions at schema load time is faulty. I have long ignored validation expression warnings when converting real applications, thinking the expressions were just invalid and ignored by Progress. This was incorrect; reviewing these now in the context of my conclusions documented in note 8 above, they (those I've looked into more closely so far) appear to be valid.

For my current test case, I see the following during schema loading:

```
WARNING: ignoring invalid option VALEXP [78:3] (error = 'line 1:1: unexpected token: udf1'):  udf1() and can-find(first person where emp-num = author-id)
```

However, I know that this is in fact a correct validation expression, which works in Progress. It's just that we're not parsing it in the context of the business logic where it will be applied.

I'm having a hard time figuring out when and how we should preprocess and inject these expressions into business logic. The same issue applies to field-level validation expressions. Currently, we are just dropping these if they do not parse at schema load time.

I guess the preprocessing step would still occur at schema load time. However, the parsing is a bit more twisted: we need to have parsed the business logic in order to have identified the locations at which to attach the validation sub-tree. However, we won't yet have a validation sub-tree at that point, since that sub-tree will need to be created by having parsed the validation expression in the context of the surrounding business logic.

Thoughts on how best to approach this?

#11 - 10/21/2015 06:42 PM - Greg Shah

Ouch. You make a good point, which I should have considered in my previous answer. Sorry about that.

Since resources in the business logic can be accessed directly in schema-defined validation expressions (and messages, I presume) then it follows that the parsing of that code cannot be done during schema loading, but must be done at the point of implicit usage in the business logic.

Here is the bad news: although we need to parse the expressions during the parser run (when we encounter a field reference in certain UI statements), we cannot dynamically insert the text into the current lexer's input. Nor can we separately lex the string read from the schema and insert those new tokens into the stream of tokens read from the lexer. There are no facilities to do such insertions and trying to create such facilities is not a good idea. It is impossible to predict the state of lexer lookahead at any arbitrary point in the parser's processing, which makes the lexer quite opaque. And although the preprocessor is already a dynamically expanding stack of cooperating streams that appear like a single stream to the lexer, it is a really complicated state machine that I don't want to break. Thus insertions at the stream level seem to be a bad idea.

And the good news: we already support nested parsing. In other words, we can create a one-off lexer and parser to process that string, during the parse of the main procedure file. In fact, we already have most of the code written for this in SchemaLoader.replaceAst().

We wouldn't be doing the replace thing, but the preprocessing/lexing/parsing is already there. I think you would have to pass in the current parser's SymbolResolver instance instead of creating a new one. That will allow resolution of scopes, variables, buffers and so forth. It won't be creating new scopes, nor will any new resources (e.g. variables) be defined since by definition we are only dealing with an expression specified in a single string literal from the schema. I wouldn't even do any preprocessing at schema loading time, just store the string literal and allow it to be retrieved.

I think the same approach will be used for both fields and for the DELETE statement. In post-parse-fixups, we currently copy these preparsed schema ASTs in schema_validations.rules, where we manufacture VALIDATION sub-tree. This manufacturing/grafting logic must be moved back to the parser. It is only safe to do this after the DELETE statement has been parsed (but before we move on to any following code. At this point, the sub-tree for the statement already exists and you can modify it/add to it. Likewise, we would process this in the associated UI statements only after a field and its format phrase has been parsed. schema_validations.rules only processes for ENABLE, PROMPT-FOR, UPDATE, SET and INSERT today. This seems incorrect because I thought that one can specify VALIDATE() wherever a format phrase can be specified. But we certainly had our reasons for implementing this way, even if I can't remember or never knew why. So for now just stick with this same approach. I would recommend creating the new sub-tree and annotating it to record that it was created artificially from the schema input. We already do that today in schema_validations.rules, you should try to duplicate the same results as much as possible so that all the downstream behavior can remain the same.

Igor: you are being copied on this because this is a big potential change to how validation expressions are processed and it will affect your current work in #2038.

#12 - 10/21/2015 08:37 PM - Constantin Asofiei

Greg/Eric: the doc for the buffer handle's BUFFER-DELETE() method states this:

> The BUFFER-DELETE method corresponds to the DELETE statement.
> If the table has delete validation—that is, if the table specifies an expression that must be true before the record is deleted—the record is not deleted, because the validation

expression, normally applied at compile time, cannot be applied fully at run time.

I don't know if this is needed for the current project, but if so, we will not be able to solve it at conversion time without hints. Parsing the validation expression at runtime I don't think is possible, as the parsed context in which this is executed is not available.

#13 - 10/21/2015 08:40 PM - Constantin Asofiei

I might have misread something: the record is not deleted, because the validation expression, normally applied at compile time, cannot be applied fully at run time. - so delete fails if there is a validation expression (it doesn't evaluate it, so that's good news). Do we track this in some meta table? Or in some other place?

#14 - 10/22/2015 02:57 PM - Eric Faulhaber

Right now, we have the original, 4GL validation expression string for the table in the _File metadata table (via the MetaFile DMO after conversion). However, this is probably not the best way to access this for this purpose. I should add this to the Java annotaions for the DMO; we do this already for field-level validation expressions.

In note 1 above, Guy specifies that dynamic deletes with a validation expressions will result in an error. So, it seems the failure to delete is not silent, but raises an error condition. I'll have to write some test cases around this to determine what that error looks like, and how uniformly this rule is applied.

#15 - 10/29/2015 02:42 PM - Eric Faulhaber

We already know SET, UPDATE, PROMPT-FOR, INSERT trigger schema-level validation. The following test results are intended to determine whether combinations of certain other UI statements trigger field-level schema validation (tests conducted with simple_bogus_database.zip from #1801):

	ENABLE ALL		VIEW W/ SENSITIVE = true	
STATEMENT	TEST CASE	RESULT	TEST CASE	RESULT
DEFINE FRAME	enable-all-def-frame.p	yes [1]	view-sens-def-frame.p	no
FORM	enable-all-form.p	yes [1]	view-sens-form.p	no
NEXT-PROMPT	enable-all-next-prompt.p	yes [1]	view-sens-next-prompt.p	no
DISABLE	enable-all-disable.p	yes [1]	view-sens-disable.p	no
DISPLAY	enable-all-display.p	yes [1]	view-sens-display.p	no
UNDERLINE	enable-all-underline.p	yes [1]	view-sens-underline.p	no
COLOR	enable-all-color.p	yes [1]	view-sens-color.p	no

Notes:

1. value must be changed from the initially displayed value (which is blank in all cases except for DISPLAY statement)

It appears to be the ENABLE statement which triggers schema-level validation, not the individual, frame-related statements. SET, UPDATE, PROMPT-FOR, and INSERT apparently involve an implicit ENABLE action.

#16 - 11/10/2015 03:25 PM - Eric Faulhaber

P2J branch 2692b, rev. 10956 (based on trunk revision 10951) contains the changes necessary to:

- parse schema validation expressions within the context of the business logic where they are used;
- add support for table-level, delete validation;
- add partial support for schema-level assign validation for the ENABLE ALL case.

The bzr log entry for the last item is misleading, in that I should have written "partial support" rather than just "support"). We attach a hidden KW_VALIDATE subtree for each field in a frame requiring validation to the STATEMENT node above the ENABLE node in this case, but further processing in downstream TRPL rules has yet to be implemented.

An earlier version of this branch (rev. 10953) passed regression testing and extensive customer unit testing.

Greg, please review.

#17 - 11/18/2015 11:12 AM - Eric Faulhaber

Improved support for ENABLE ALL is now in 2692b/10961. This is a release candidate which I am testing currently.

#18 - 11/22/2015 10:07 AM - Greg Shah

Code Review Task Branch 2692b Revision 10961

This is really good work. It is very close to complete.

1. The frame_scoping.rules on line 4500 probably has the unintended consequence of calling add_widget twice for the ENABLE ALL EXCEPT case. Since isWidget is now false, the add_widget will be called on lines 4511 and 4539. This may be the source of the funky widget naming changes you saw. I suspect at least some of them are caused by this. Not sure, but one solution might be to set isWidget to false on line 4512.

2. Please check if the VALMSG can be an complex expression instead of a literal. In business logic, the VALMSG is definitely parsed as a full expression. If a complex expression is possible, then attachSchemaValidation() should parse the VALMSG text too. I realize that the old SchemaLoader.replaceAst() implementation did not do this. That may have been another bug.

3. I'm surprised that in record_spec, you did not exclude the fields in except_list.

4. An enable_stmt with a field_clause needs to handle the normal addFrameFields() processing, just like the disable_stmt.

5. I don't understand why insert_stmt and set_or_update_or_prompt_for_stm don't pass a set into record_spec. Those statements may be the frame defining statement for a subsequent ENABLE ALL. It doesn't seem correct to treat them differently than a DISPLAY, for example.

6. The set_or_update_or_prompt_for_stm case supports a non-record case where explicit fields are added via chained_object_members. This needs support too.

7. I wonder if we will need to do the same thing to honor fields/records specified in browse columns too. We currently don't support validation in browse columns, but we will be changing that in the coming months. Please put a TODO in the column_spec rule to highlight that fact.

#19 - 11/22/2015 10:08 AM - Greg Shah

I've committed branch 2692b rev 10962 with some minor comment and formatting cleanups.

#20 - 11/23/2015 01:09 PM - Eric Faulhaber

Greg Shah wrote:

> 6. The set_or_update_or_prompt_for_stm case supports a non-record case where explicit fields are added via chained_object_members. This needs support too.

Would you please post a short 4GL example (just the relevant SET/UPDATE/PROMPT-FOR line with the chained object member(s)) which shows this case?

#21 - 11/23/2015 01:10 PM - Greg Shah

```
UPDATE tt.num tt.num2 WITH FRAME whatever.
```

#22 - 11/23/2015 01:17 PM - Greg Shah

Any database field or lvalue can be inserted instead of the temp-table fields. The point is that these statements are the common case. chained_object_members is essentially handling the lvalue here.

#23 - 11/23/2015 01:56 PM - Eric Faulhaber

Greg Shah wrote:

> 2. Please check if the VALMSG can be an complex expression instead of a literal. In business logic, the VALMSG is definitely parsed as a full expression. If a complex expression is possible, then attachSchemaValidation() should parse the VALMSG text too. I realize that the old SchemaLoader.replaceAst() implementation did not do this. That may have been another bug.

I checked this previously, and I think I had checked way back when we did the SchemaLoader implementation as well. Any expression entered in the data dictionary for a validation message is just printed as is; it is not evaluated. I'm pretty sure everything is just treated as a string.

#24 - 11/23/2015 01:58 PM - Eric Faulhaber

Greg Shah wrote:

> 3. I'm surprised that in record_spec, you did not exclude the fields in except_list.

This was a bit complicated to do in the parser and it didn't seem necessary, since the exception list is taken into account during record field expansion downstream in TRPL.

#25 - 11/23/2015 03:29 PM - Greg Shah

3. I'm surprised that in record_spec, you did not exclude the fields in except_list.

> This was a bit complicated to do in the parser and it didn't seem necessary, since the exception list is taken into account during record field expansion downstream in TRPL.

OK. Please put a comment there to explain this.

#26 - 11/24/2015 06:56 AM - Greg Shah

Please rebase to trunk 10952.

#27 - 11/25/2015 07:16 AM - Greg Shah

Eric rebased task branch 2692b from P2J trunk revision 10952. The latest revision is now 10964.

#28 - 11/25/2015 07:17 AM - Greg Shah

Code Review Task branch 2692b Revision 10964

The changes are good.

#29 - 11/25/2015 07:18 AM - Greg Shah

Please provide a status update on your conversion and runtime regression testing.

Also, please write up a summary of your findings regarding the 4GL schema validation behavior.

#30 - 11/25/2015 10:39 AM - Eric Faulhaber

I'm going to call conversion regression testing passed. There are some differences in some frames' widget method names, but they are consistent across frame definition and business logic, and the code compiles. The change to frame_scoping.rules line 4512 suggested above did not change this. No other differences in Majic code.

Differences in server and gui converted code are all expected and look correct, and the code compiles.

The schema loading step is of course much cleaner now. There are a small number of parser warnings, due to incorrect schema validation expressions. These are expected, and are due to the fact that we don't take the NO-VALIDATE option on a frame into account until after the parsing phase, so we parse all schema validation expressions we find in a frame definition. This is actually a useful side effect, as it provides a record of invalid validation expressions which have managed to slip into a schema, even if they are never actually invoked in an application.

Majic runtime testing main regression had a handful of errors; I'm re-running now. Haven't run ctrl-c tests yet, but I don't expect any surprises there,

based on my changes. Unit testing is in process.

#31 - 11/25/2015 10:49 AM - Greg Shah

> This is actually a useful side effect, as it provides a record of invalid validation expressions which have managed to slip into a schema, even if they are never actually invoked in an application.

Actually, they could be successfully used elsewhere in the application. In all likelihood, the addition of the NO-VALIDATE was made to avoid the compile failure in these specific cases. If it was not needed elsewhere, then the developers could have removed the expression from the schema.

#32 - 11/25/2015 05:18 PM - Eric Faulhaber

Runtime main and ctrl-c regression testing passed. Server unit testing (full search, common search, system maintenance, developer tests) showed no regressions.

I've merged 2692b/10964 to trunk revision 10953.

#33 - 12/01/2015 08:17 AM - Greg Shah

Can we close this?

#34 - 12/01/2015 10:19 AM - Eric Faulhaber

Greg Shah wrote:

> Can we close this?

All the work with the code is done and merged to trunk. I just left this open to document more what was done and why. Will do that shortly, then close it.

#35 - 03/10/2016 11:07 AM - Eric Faulhaber

One other aspect of this task is still open: I've discovered that our implementation of the BUFFER-DELETE() method is incomplete. If the table containing the target record has schema-level delete validation, the BUFFER-DELETE() method must fail with the error:

```
Table <table name> is defined with schema delete validation and may only be deleted by the DELETE statement. (
7347)
```

LE: this bug will be managed separately, under issue #3026.

#36 - 04/29/2016 12:54 AM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from WIP to Closed

The essence of adding this feature was to move all the table-level validation expression parsing from the SchemaLoader (which was too early) to the Progress parser (to have the benefit of business logic context, since validation expressions can reference resources in this context).

Along the way, we discovered that the same had to be done for all assign (i.e., field-level) validation expressions, which increased the scope of the task considerably. As part of this, all post-parse schema validation work which attached the assign validation expression and message AST nodes created by the SchemaLoader to ENABLE, PROMPT-FOR, SET, UPDATE, and INSERT statement nodes was removed/refactored into the parser. The parsing work previously done (incorrectly) during schema loading is now done during main parsing, by using a nested parser as described in note 11 above.

Frames need special handling, because they can contain field widgets with schema validation, whose validation is triggered with ENABLE ALL. The SymbolResolver was modified to keep track of the field widgets referenced by a frame, so the parser could attach validation nodes to these widgets when parsing the ENABLE ALL statement. Any of the statements in note 15 (DEFINE FRAME, FORM, NEXT-PROMPT, DISABLE, DISPLAY, UNDERLINE, COLOR) can associate a field with the frame, such that a subsequent ENABLE ALL (or PROMPT-FOR, SET, UPDATE, INSERT) statement will trigger validation. Thus, when any of those associating statements are encountered during parsing, they register the related field with the current frame in the symbol resolver.

The parser now attaches table-level (delete) schema validation nodes to DELETE statements (unless there is an explicit VALIDATE statement, which pre-empts the schema validation). It also attaches field-level (assign) schema validation nodes to ENABLE, PROMPT-FOR, SET, UPDATE, and INSERT statements (unless there is an explicit VALIDATE statement, which pre-empts the schema validation). When ENABLE ALL is encountered (and no explicit VALIDATE statement), the parser uses the symbol resolver to look up all the field ASTs which have been registered with the related frame by the associating statements. For each such field, a schema validation node is attached to a new format phrase node attached beneath the ALL AST.

From this point, downstream TRPL processing emits the necessary validation inner classes and registers them with the appropriate widgets.

Note that Stanislav today found a bug in the delete validation, where the validation message was not being attached properly. I have made fix for this and the patch is posted in #3026. He will roll the fix into his updates for that task.

#37 - 04/29/2016 08:11 AM - Greg Shah

There are some things I would like to clarify. Please correct anything I get wrong here.

1. It is not just ENABLE ALL that triggers the copying of validation expressions from the schema. Any ENABLE statement does this and we support that now at the parser level.

2. The toughest problem that had to be solved was that the ENABLE ALL has an implicit widget list. Prior to these changes, the P2J parser didn't track its widgets on a per frame basis. Figuring out which widgets went into which frame was done later in conversion. The changes for this feature had to track all frame defining statements and put any field references into the list for schema validation copying.

3. There was another special case that made this ENABLE ALL implicit widget list difficult to track: the record reference case (DISPLAY my-record WITH FRAME some-frame.). In the 4GL code this is the equivalent of listing all fields in the record. So for this case, we had to add all the fields from the record into the frame-level tracking for validation purposes.

#38 - 04/29/2016 08:15 AM - Greg Shah

Questions:

- I recall that validation expressions be copied in from the schema more than once (if there are multiple ENABLE statements). Only 1 survives to be used in the resulting frame, if I remember right. What were the rules that determined which one is kept?
- Where did you store the testcases that were created to show all this behavior?

#39 - 04/29/2016 03:57 PM - Eric Faulhaber

Greg Shah wrote:

> There are some things I would like to clarify. Please correct anything I get wrong here.
>
> 1. It is not just ENABLE ALL that triggers the copying of validation expressions from the schema. Any ENABLE statement does this and we support that now at the parser level.

Correct, as noted above.

> 2. The toughest problem that had to be solved was that the ENABLE ALL has an implicit widget list. Prior to these changes, the P2J parser didn't track its widgets on a per frame basis. Figuring out which widgets went into which frame was done later in conversion. The changes for this feature had to track all frame defining statements and put any field references into the list for schema validation copying.

Yes.

> 3. There was another special case that made this ENABLE ALL implicit widget list difficult to track: the record reference case (DISPLAY my-record WITH FRAME some-frame.). In the 4GL code this is the equivalent of listing all fields in the record. So for this case, we had to add all the fields from the record into the frame-level tracking for validation purposes.

Yes, the fields are collected in the record_spec rule of the parser, if we pass an optional fields set into that rule.

#40 - 04/29/2016 04:20 PM - Eric Faulhaber

Greg Shah wrote:

- I recall that validation expressions be copied in from the schema more than once (if there are multiple ENABLE statements). Only 1 survives to be used in the resulting frame, if I remember right. What were the rules that determined which one is kept?

I don't recall implementing this. AFAIR, any ENABLE statement with a field list or an ALL keyword triggers the schema validation to be attached.

- Where did you store the testcases that were created to show all this behavior?

uast/schemaval

#41 - 11/16/2016 12:12 PM - Greg Shah

- Target version changed from Milestone 12 to GUI Support for a Complex ADM2 App

#42 - 08/14/2017 05:11 PM - Greg Shah

- Related to Feature #3281: implement some frame options added