User Interface - Bug #3575

"implicit copy from screen buffer" version of the ASSIGN statement is broken

05/17/2018 05:05 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	
Description			

History

#1 - 05/17/2018 05:43 PM - Greg Shah

This was originally identified as a compile-time failure in #3567-23 (see that note and most of the rest up to #3567-50).

The ASSIGN statement normally has a list of explicit assignment expressions in the form lvalue = expr. There is an implicit copy back from screen-buffer mode where there is no = expr specified. In such a case, where one or more lvalues is listed without any rvalue or assignment operator, the 4GL docs say that the frame or browse must be specified. In fact, it is optional.

For many cases, it will probably be common that the lvalues listed only could come from a single screen-buffer and so the lack of qualifier doesn't matter.

In #3567 it was found that the 4GL has some precedence rules that can choose between different screen-buffers. Multiple variants of the following testcase were tried and the rules are not yet clear.

```
def temp-table tt
   field f1 as int
   field f2 as int.

def query qtt for tt.

def browse brws query qtt
   display tt.f1 tt.f2
   enable tt.f1 tt.f2
  with 5 down.

form brws with frame fbrws.
form tt.f1 with frame f-partial-1.

create tt.
tt.f1 = 10.
tt.f2 = 20.

open query qtt for each tt.

/* CASE 1: type 1 TAB 2 F2 */
/* CASE 2: type 1 F2 */
prompt-for brws with frame fbrws.
assign tt.f1 tt.f2. /* CASE 1 and 2: targets fbrws */
message "A" tt.f1 tt.f2.
/* CASE 1: shows "A 1 2" */
/* CASE 2: shows "A 1 20" */

/* CASE 1 and 2: type 3 F2 */
prompt-for tt.f1 with frame f-partial-1.
assign tt.f1 tt.f2. /* CASE 1 and 2: this targets frame f-partial-1 */
message "B" tt.f1 tt.f2.
/* CASE 1: shows "B 3 2" */
/* CASE 2: shows "B 3 20" */

/* CASE 1 and 2: type 4 F2 */
prompt-for brws with frame fbrws.
```

```
assign tt.f1 tt.f2.  /* CASE 1: this targets frame fbrws; CASE 2: this targets f-partial-1 */
message "C" tt.f1 tt.f2.
/* CASE 1: shows "C 3 4" */
/* CASE 2: shows "C 3 20" */
```

FWD currently makes the determination of which frame is being processed at conversion time. This led to code that could not compile, which is what brought this case to our attention.

In fact, most of the testing suggested that the 4GL makes the decision at compile time. This can be seen is you run the COMPILE statement on code which can cause compile warnings. For example, if it detects that there is a PROMPT-FOR that has no matching ASSIGN, you will see the non-fatal warning ** PROMPT variable f1 should be used with INPUT prefix or ASSIGNed. (402). This happens during COMPILE, not at runtime. Even if the PROMPT-FOR exists in a block that can never be executed (e.g. IF FALSE THEN DO: /* failing code */ END.) the compile warning will occur. This makes some sense because all static references to frame fields must be determined at compile time since that is the time at which the frame definition is calculated.

But the testcase above seems to show that to some degree the choice of frame is actually made at runtime and seems to be affected by which fields get edited. In other words, different user input for the same testcase results in copying back from different frames.

I suspect that what matters is the intersection of fields that are listed (in the implicit ASSIGN) and the fields that were actually edited in a recent interactive editing session.

1. Early testing suggests that the order in which non-referencing frame statements (e.g. FORM) appear does not seem to make a difference. This would make sense if this behavior is runtime based since those statements don't actually execute, they just help the compiler define the static frame.
2. The decision is made at each ASSIGN statement, so it will change depending on what frame referencing statements have been executed before that point. It is not clear to what degree conditional processing matters, but presumably if this is runtime based then conditional control flow will have a big impact.
3. The 4GL seems to silently drop any non-matching fields from the implicit copy back. FWD will fail compilation on these.
4. The 4GL will not copy back from fields that have not been edited. I don't know if FWD handles this or not, but I seem to recall we do handle it.
5. There seems to be a preference for "direct frame widget references" over browse column references. We can see that a field in the ASSIGN only exists as a browse column and **not in any other frames that have a referencing statement UP TO THAT POINT**, then that browse's frame will be chosen. But as soon as any frame directly contains (not in a browse, but as a direct widget) the field, then for the rest of the program the browse frame cannot be implicitly selected for an ASSIGN. There are many open questions:
    - Do the number of matching fields matter?
    - Does the order of the matching fields matter?
    - Does a partial match (some fields match but other fields are not specified) have different precedence than a full match (all fields are specified and match)?
    - Does the level of partial match make a difference. For example, if one frame has 2 of the fields that match the assign and another has 1 field that matches, will the 2 field frame always "win"? Or is there some kind of ordering like the most recent reference wins or the first one executed wins?
    - If 2 frames have the same fields referenced but the order is different, is there a difference in behavior?
    - If the 2 ASSIGN statements are the same except for the order of the fields, is there a difference in behavior?
    - Does the inclusion of non-existing fields change the behavior? We know it is possible to match a frame that doesn't contain all of the references in the implicit assign, but we don't know how the non-existing fields affect things.
    - Does the type of referencing statement matter? In particular, PROMPT-FOR may have special significance since the ASSIGN is needed to copy back changes (SET and UPDATE handle the ASSIGN internally).
    - Check if variables act the same way as fields.
    - Does scope matter?

If this is indeed a runtime thing, then a very different conversion will be needed.

#3 - 05/18/2018 01:46 PM - Constantin Asofiei

Note that in the original program which showed this issue the ASSIGN was enclosed in a EDITING block for an UPDATE statement.