Conversion Tools - Support #11878

gap marking updates

09/15/2026 02:57 PM - Greg Shah

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	90%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

History

#1 - 09/15/2026 02:59 PM - Greg Shah

- File gap_marking_cleanup_1_11747a.diff added

- Status changed from New to Review

- % Done changed from 0 to 50

Please review these proposed updates. I found them via scanning using Claude and plan to commit them into 11747a unless someone finds a problem.

#2 - 09/15/2026 05:14 PM - Greg Shah

I committed these changes as 11747a revision 16752, so that I can work on other changes in the meantime. We can make any changes needed directly in that branch.

#3 - 09/15/2026 06:15 PM - Ovidiu Maxiniuc

I have checked the following database-related elements and they seem correct in the patch/branch:
_field._field-physpos,
_myconnection._myconn-numseqbuffers,
_myconnection._myconn-usedseqbuffers,
_myconnection._myconn-tenantid,
prog.kw_col_cp: OK in ttFieldOpts; should be deleted from opts
prog.kw_is_colcp: both whereFuncs and funcs are OK,
prog.kw_rejected,
prog.kw_auto_del,
prog.kw_ign_cmod,
prog.kw_mark_rs,
prog.kw_min_schm,
kw_no_sch_m,
prog.kw_no_undo: ttOpts OK; opts not sure what statements are affected: DEFINE PARAMETER, DEFINE VARIABLE, etc
prog.kw_ttcp: ttOpts OK; should be deleted from opts (same as kw_col_cp)

#4 - 09/16/2026 06:08 AM - Greg Shah

Ovidiu Maxiniuc wrote:

> I have checked the following database-related elements and they seem correct in the patch/branch:
> _field._field-physpos,
> _myconnection._myconn-numseqbuffers,
> _myconnection._myconn-usedseqbuffers,
> _myconnection._myconn-tenantid,
> prog.kw_col_cp: OK in ttFieldOpts; should be deleted from opts
> prog.kw_is_colcp: both whereFuncs and funcs are OK,
> prog.kw_rejected,
> prog.kw_auto_del,
> prog.kw_ign_cmod,
> prog.kw_mark_rs,
> prog.kw_min_schm,
> kw_no_sch_m,
> prog.kw_no_undo: ttOpts OK; opts not sure what statements are affected: DEFINE PARAMETER, DEFINE VARIABLE, etc
> prog.kw_ttcp: ttOpts OK; should be deleted from opts (same as kw_col_cp)

Please apply any of your recommended deletions to the branch.

#5 - 09/16/2026 06:59 AM - Constantin Asofiei

Greg, I've looked at the changes (non-db), and I can't find anything else to contradict.

#6 - 09/19/2026 06:35 AM - Greg Shah

UI Object Inventory Reports

Introduction

FWD Analytics had no answer to the simplest questions a reader asks about a converted application's user interface: how many frames does it have, how many windows, how many dialogs, and what widgets does it put on the screen. Every existing User Interface report counts *language feature usages* — how many DISPLAY statements, how many frame options, how many UI events. None of them counts *objects*.

This note records why that gap could not be closed with a report definition alone, the analysis of what stood in the way, and the design and implementation of the change that closed it.

The work was committed to the 11747a branch as revision 16763, referencing #11746. It adds one new rules file and modifies four others. No conversion behaviour changes.

The requirement

Five reports were wanted, all in the User Interface section.

Report	One match per	Categorized by
Dynamic Widget Usage	widget creation which cannot be named in 4GL	widget type

	source	
Static Widget Usage	widget instance placed in a frame	widget type
Frames	unique frame in the application	frame name
Windows	unique window in the application	how the window comes into existence
Dialogs	unique dialog in the application	the 4GL feature which produces it

The phrase "one match per unique object" is what makes these different from every report that came before, and it is the entire source of the difficulty.

Problem analysis

The report engine has one match per AST node

A report in rules/reports/profile.rpt is a condition evaluated at every node of every parsed tree, a multiplexExpr which names the category a matching node falls into, and a dump expression. One matching node produces exactly one match row. There is no aggregation step and no notion of identity beyond the node.

For a report which counts language features this is exactly right — a DISPLAY statement is one usage of DISPLAY. For a report which counts objects it is wrong, because a 4GL object is not a node.

A 4GL frame is not a node

A frame can be built up by any number of statements:

```
DEFINE FRAME fMain
   cName   AT ROW 2 COLUMN 5
   cChoice AT ROW 3 COLUMN 5
   WITH SIDE-LABELS THREE-D.

FORM
   lFlag  AT ROW 4 COLUMN 5
   cNotes AT ROW 5 COLUMN 5
   WITH FRAME fMain.

ENABLE cName cChoice WITH FRAME fMain IN WINDOW hWin.
```

That is one frame, four widgets, and three statements which all refer to it. A report condition matching DEFINE_FRAME would miss frames that have no DEFINE; a condition matching FRAME_PHRASE would count this frame twice; a condition matching frame name references would count it three times. The unnamed frame is worse still — it has no name to key on, and each block that uses one has its own.

Deciding which references aggregate into which frame is frame scoping, and it is not a local property of any node. It needs a walk that tracks block scopes, a per-procedure registry of named frames, and rules for how an inner block's frame merges outward.

Frame scoping already exists, and runs far too late

rules/annotations/frame_scoping.rules does exactly this calculation. It cannot be used:

- It runs from annotations.xml, which executes in the **middle** of the conversion, after record scoping, where-clause processing and copy_view_as_from_schema. Analytics reads the ASTs the **front end** leaves behind. TransformDriver.front() runs the scan driver, then Post-Parse Fixups, then Early Annotations, then Gap Analysis Marking, and stops. ant rpt-no-front starts from that point.
- Pointing analytics at fully annotated trees instead is not an option. annotations.xml removes unreachable code, expands arrays, rewrites where clauses and reparents statements. Every existing report would silently change meaning.

Frame scoping cannot cheaply be moved or split

frame_scoping.rules is 6,303 lines in a single rule-set: roughly 4,250 lines of functions, 1,170 lines of descent rules, 240 of ascent rules. Scope resolution is interleaved line by line with two things that are illegal in the front end:

- **Java name generation.** p2o_name, gen_accessors, gen_fld_getter, put_accessor and their callers produce the converted field and widget accessor names.
- **Tree modification.** It creates FRAME_SCOPE and FRAME_ALLOC nodes, hides subtrees, and retypes CURRENT-WINDOW assignments. process_frame, which does most of this, is 860 lines on its own.

early_annotations.xml carries an explicit prohibition at the top of the file:

```
DO NOT MAKE TREE CHANGES WHICH ARE ONLY USEFUL FOR CONVERSION; ALL CHANGES MUST
BE SUITABLE FOR REPORTING OR ANALYSIS PURPOSES SIMILAR TO THE LIMIATIONS IN
POST-PARSE FIXUPS
```

Splitting the file into a scope-analysis core and a codegen tail, and making the late pass consume the early pass's results, is the eventual right answer. It is also a deep refactor of a twenty-year-old file where every regression lands in converted output.

What the other four reports need

Windows and Dialogs are only partly a frame problem.

- A window is created by CREATE WIDGET with a WINDOW type, which is one statement and needs no scoping. The implicit DEFAULT-WINDOW is the hard half: it exists whenever a frame is realized and no window was explicitly given, which needs to know what a frame is.
- A dialog comes from four different features. CREATE WIDGET with DIALOG-BOX, MESSAGE ... VIEW-AS ALERT-BOX and the five SYSTEM-DIALOG statements are each one statement and one dialog. The fourth, a frame whose FRAME_PHRASE carries VIEW-AS DIALOG-BOX, is a frame and inherits the whole problem.

Static Widget Usage needs, for every widget, the effective widget type. A field or variable placed in a frame is a FILL-IN unless a VIEW-AS applies, and that VIEW-AS can come from the schema, the temp-table definition, the variable definition, or the format phrase written in the frame-defining statement itself.

Dynamic widget creation, and one thing that was nearly missed

CREATE WIDGET and CREATE BROWSE are the only statements that create a widget which no 4GL statement can afterwards name. The grammar rule create_widget_or_object_stmt in progress.g takes 31 widget keywords, a VALUE(expression), or a bare expression; the TIMER, SMTP-EMAIL, REPORT and COM-handle forms are retyped to CREATE_TIMER, CREATE_SMTP_EMAIL, CREATE_REPORT and CREATE_OBJECT during the parse, so CREATE_WIDGET needs no filtering.

Three browse handle methods were nearly overlooked. Each creates one or more browse column widgets at run time and returns a handle:

Method	Token
ADD-LIKE-COLUMN	KW_ADD_L_C
ADD-CALC-COLUMN	KW_ADD_C_C
ADD-COLUMNS-FROM	KW_ADD_C_F

They are already gap marked in rules/gaps/expressions.rules but were reported nowhere as widget creation. They belong in Dynamic Widget Usage for exactly the reason CREATE WIDGET does: the widgets they create have no name.

Two related cases were considered and deliberately left out. chCtrlFrame:LoadControls() instantiates an ActiveX control inside a control frame, but that is a COM object and already has its own report family. The transient dialogs raised by MESSAGE ... VIEW-AS ALERT-BOX and SYSTEM-DIALOG are runtime-created widgets with no handle, and are counted as dialogs rather than as dynamic widgets.

Design

The shape of the solution

Exactly one AST node per unique object is marked with an annotation during the front end, and each report's condition tests for that annotation. The report engine's one-node-one-match model is then correct without modification.

Four decisions were taken before implementation.

1. **A new, purpose-built early pass**, rather than refactoring or sharing frame_scoping.rules. It computes only what the reports need and touches nothing the conversion depends on, so the regression risk to converted output is zero. The cost is a second implementation of the scope algorithm, which is mitigated by the cross-check described below.
2. **DEFAULT-WINDOW is counted once per external procedure** that realizes a frame with no explicit window. It overstates the number of windows — there is only ever one DEFAULT-WINDOW per session — but it shows which programs depend on it, which is what a gap document needs.
3. **A static widget is one widget instance in one frame.** A field placed in two frames counts twice. This matches 4GL runtime reality and reuses the per-frame widget lists the pass has to build anyway.
4. **The Frames report multiplexes on the frame name**, with the unnamed frame of each block collected under a single category.

The new rule set

rules/annotations/frame_identity.rules, 1,089 lines, invoked from early_annotations.xml after com_origin and before the persist. It is annotation-only and modifies no node.

It reimplements a reduced form of the frame_scoping.rules machinery, keeping the parts that decide identity and discarding everything else:

Kept	Discarded

procScope, a stack of named-frame registries, one per external procedure, internal procedure, function and trigger	Java name generation: p2o_name, gen_accessors, gen_fld_getter, put_accessor
A dictionary scope per frame-scoping block, holding that block's unnamed frame	FRAME_SCOPE and FRAME_ALLOC node creation
A temporary scope per frame-referencing statement, committed outward on ascent	DOWN and frame-type tracking
Widget collection into the frame in scope	Subscript and array accessor handling
Merging a statement's frame into the enclosing block's frame, or into the named frame it declared	Browse cross-referencing, "at base" clause rewriting

Two predicates had to be duplicated rather than reused. is_block and is_disp are private to frame_scoping.rules, and common-progress.rules has a **different** is_block(target) which answers an unrelated question. They appear here as is_frame_block and is_frame_disp.

The annotations it leaves behind

Annotation	Type	Meaning
frame-primary	boolean	exactly one node per unique frame carries this
frame-key	String	<relative path>#<n>, unique across the application
frame-legacy-name	String	legacy frame name, or empty for the unnamed frame
frame-unnamed	boolean	true for the unnamed (default) frame of a block
frame-dialog	boolean	the frame is realized as a dialog box
frame-kind	String	how the frame is introduced
frame-window	String	the IN WINDOW expression, when one was given
frame-widgets	long	number of distinct static widgets in the frame
frame-widget-of	String	frame-key of the frame this widget belongs to
frame-widget-type	String	4GL widget type, for example FILL-IN, BUTTON, BROWSE
uses-default-window	boolean	set on the root BLOCK of a compilation unit which realizes a frame with no explicit window

The name frame-legacy-name is deliberate. frame-name was the obvious choice and is already used as an AST annotation by rules/convert/frame_generator.xml, so it was avoided.

Resolving the widget type

resolve_widget_type applies the 4GL precedence directly:

1. A VIEW-AS in the format phrase attached at this placement wins.
2. Otherwise a VIEW-AS on the variable or temp-table field definition, reached through the node's refid annotation.
3. Otherwise a reference to a widget carries its own type.
4. Otherwise a literal is display-only text, reported as LITERAL.
5. Otherwise the widget is a FILL-IN. SKIP and SPACE are layout and are not widgets.

Determining the window

A frame is associated with a window by an IN WINDOW clause, which appears either in the frame phrase or directly in a statement such as VIEW FRAME f IN WINDOW w. Both forms are handled by resolving the frame the clause applies to from the FRAME option in the same statement, falling back to the frame currently in scope.

Where no frame names a window, the compilation unit is using the DEFAULT-WINDOW — unless it assigns CURRENT-WINDOW, which re-parents every frame realized afterwards. The AppBuilder pattern makes this the common case:

```
CREATE WINDOW C-Win ASSIGN ...
ASSIGN CURRENT-WINDOW = {&WINDOW-NAME}.
```

Assigning DEFAULT-WINDOW to CURRENT-WINDOW is the exception and leaves the default in place.

A dialog box counts towards the DEFAULT-WINDOW. 4GL parents a dialog on the current window just as it does a plain frame, so a program whose only UI is a dialog still depends on the DEFAULT-WINDOW being there.

Widgets with a DEFINE of their own

DEFINE BROWSE, DEFINE BUTTON, DEFINE IMAGE and DEFINE RECTANGLE declare a widget which is normally placed in a frame and counted there. One that is never placed would otherwise vanish from the report entirely, so the pass remembers every such definition and every widget name actually placed, and annotates the definitions left over at the end of the compilation unit.

Implementation

Files

File	Change
rules/annotations/frame_identity.rules	new, 1,089 lines
rules/annotations/early_annotations.xml	invoke the new rule set
rules/include/common-progress.rules	add widget_type_name
rules/include/report.rules	add the report predicates and multiplex helpers
rules/reports/profile.rpt	add the five reports

widget_type_name

A widget type has to be named from its token rather than from the node's text, because 4GL keywords may be abbreviated — CREATE FI h writes FI. widget_type_name in common-progress.rules maps the keyword tokens, the WID_* reference tokens and the DEFINE_* statement tokens to one canonical 4GL name each. It lives in common-progress.rules because both the annotation pass and the report helpers need it.

The report helpers

rules/include/report.rules gained a predicate and a describe function per report. The three that read annotations are trivial; the two that do real work are:

- dynamic_widget_creation / describe_dynamic_widget — matches CREATE_WIDGET, CREATE_BROWSE and the three browse column methods. For CREATE_WIDGET it unwraps VALUE(...) and expression wrappers; a string literal names the widget type as well as a keyword does and is reported under that type, and everything else falls into a single runtime category.
- static_widget / describe_static_widget — matches anything carrying frame-widget-type, every frame (reported as FRAME or DIALOG-BOX), and the menu structures, which are never frame elements.

The reports

```
<report
   condition="evalLib(&amp;quot;dynamic_widget_creation&amp;quot;, this)"
   dumpType="parser"
   multiplexExpr="execLib(&amp;quot;describe_dynamic_widget&amp;quot;, this)"
   supportLvlExpr="execLib(&amp;quot;read_support_level&amp;quot;, this)"
   title="Dynamic Widget Usage"
   tags="User Interface"/>
```

Dynamic Widget Usage carries a support level because every node it matches is already gap marked. The other four are inventory reports and carry no supportLvlExpr and no featureType, so they do not feed the gap analysis document — which is correct, since their categories are application names rather than 4GL features.

Verification

Cross-check against frame scoping

The pass is a second implementation of an algorithm that already exists, so the two must be shown to agree. After a full ant convert, frame_scoping.rules has created one FRAME_ALLOC node per frame it found. Counting those and counting frame-primary annotations in the same trees gives, per file:

File	FRAME_ALLOC	frame-primary
UiInventory.p	3	3
UiDefaultWindow.p	2	2

Test beds

Two projects were used. poc is a customer application of 67 parsed trees including AppBuilder GUI code, taking about four minutes for ant clean.convert rpt. gaptest is a minimal four-file analytics project which runs in seconds and was extended with two synthetic programs, UiInventory.p and UiDefaultWindow.p, covering the paths no real code in the test bed exercises: CREATE VALUE() with a literal and with a variable, all three browse column methods, menus, a frame split across statements, a format phrase overriding a definition's VIEW-AS, and a program which relies on the DEFAULT-WINDOW.

Results

poc:

Report	Categories	Matches
Dynamic Widget Usage	5	33
Static Widget Usage	10	91
Frames	11	18
Windows	2	10
Dialogs	3	162

gaptest:

Report	Categories	Matches
Dynamic Widget Usage	16	16
Static Widget Usage	14	24
Frames	4	5
Windows	2	2
Dialogs	8	9

Independently confirmed on poc: the 33 Dynamic Widget Usage matches are exactly the 33 CREATE_WIDGET and CREATE_BROWSE nodes in the project, and the Static Widget Usage FRAME and DIALOG-BOX categories sum to 18, the total number of frames.

All Dynamic Widget Usage categories report a support level of 16400, CVT_LVL_FULL | RT_LVL_FULL. No category in any of the five reports is Unknown.

Two defects found while verifying

A later reference overwrote the defining statement. Merging one frame model into another used putAll on the widget map. Because the map is keyed by widget name, a later DISPLAY cList WITH FRAME fDialog replaced the node from DEFINE FRAME fDialog, and with it the format phrase. A VIEW-AS FILL-IN written in the frame definition was silently lost and the variable's own VIEW-AS SELECTION-LIST won instead. The first statement to place a widget in a frame is the one that defines it, so the merge now keeps the first occurrence.

Menus and unplaced widgets reported as Unknown. DEFINE_MENU and DEFINE_SUB_MENU had no entry in widget_type_name, and a widget never placed in a frame was not reported at all. Both are fixed; gaptest now reports MENU, SUB-MENU, MENU-ITEM and the unplaced DEFINE BROWSE.

Limitations and open items

- **Not run on a large project.** poc is 67 trees and gaptest is six. Neither exercises the volume of a customer application.
- **ADD-LIKE-COLUMN and friends are categorized by method, not by widget type**, as BROWSE COLUMN (ADD-LIKE-COLUMN) and so on. ADD-LIKE-COLUMN can be given the type as a literal argument and could be refined; the other two cannot be known statically.
- **All non-literal CREATE VALUE() usages share one category.** The widget type is genuinely unknown until run time, so there is nothing more specific to say, but a per-expression breakdown is possible if wanted.
- **Frame scoping is still duplicated.** Folding frame_scoping.rules onto these annotations remains the eventual goal, and is now possible because the annotations it would consume are already on the tree when it runs.
- **Two pre-existing conversion failures** were found while cross-checking and are unrelated to this work. poc fails a full ant convert on sysinf01.w with "null value for annotation 'containing-package'" raised from oo_references.rules for System.IO.DriveInfo; gaptest fails on DotNetEnum.p with "null value for annotation 'legacy-import'". A control run with the new rule set disabled reproduces the poc failure exactly. ant clean.convert rpt succeeds on both projects.

#7 - 09/19/2026 06:38 AM - Greg Shah

- % Done changed from 50 to 90

The new UI reports is my attempt to provide a better sense of the number/scale of the screens in a given application. I also added UI metrics to the gap analysis report.

Hynek: Please review the UI reporting improvements. I'm especially wanting your review of the logic that calculates the frames, windows and dialogs.

#8 - 09/20/2026 02:37 AM - Hynek Cihlar

Greg Shah wrote:

> Hynek: Please review the UI reporting improvements. I'm especially wanting your review of the logic that calculates the frames, windows and dialogs.

Two points.

1. dialog-box, alert-box and system-dialog should not collapse into a single report entity - dialog. They are different elements from the POV of UI and 4GL as well.

2.

```
DEFINE BROWSE, DEFINE BUTTON, DEFINE IMAGE and DEFINE RECTANGLE declare a widget which is normally placed in a
 frame and counted there. One that is never placed would otherwise vanish from the report entirely, so the pas
s remembers every such definition and every widget name actually placed, and annotates the definitions left ov
er at the end of the compilation unit.
```

If a widget is declared, but not used in any frame, should it be counted? Isn't this just dead code?

Files

gap_marking_cleanup_1_11747a.diff	66.4 KB	09/15/2026	Greg Shah