Conversion Tools - Feature #11860

store and report gap details in analytics

09/10/2026 01:59 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
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 - 09/10/2026 02:07 PM - Greg Shah

We need to add a facility to store the details of gaps that are reported (useful detail can be found in Gap Analysis). All support levels could possibly need to record some details, except for CVT_LVL_FULL and RT_LVL_FULL, which both imply there is no limitation. Since there are 2 separate values recorded (for conversion and for runtime), we need to be able to store the 2 sets of gap details (one for conversion and the other for runtime).

- Where the gap marking data comes from rule-sets, the details probably also need to be in the rule-sets.
- Where the gap marking data comes from Java class annotations, the details need to be included in those annotations.
- It is perfectly fine to have no gap details recorded (most cases will be that).
- In some cases, we have followed a convention where in a rule-set, where we have a line that specifies support levels, we often will have a comment at the end of the line with some notes about gaps). For example, <rule>attrs.put(prog.kw_get_iter, rw.cvt_lvl_full | rw.rt_lvl_partial)</rule> <!-- runtime is fully supported for FRAME widget but has only stub for BUFFER object --> from rules/gaps/expressions.rules. Some of these may be out of date, some may be missing or cryptic. These are only sometimes useful but where they exist we should move the text into the new facility.

#2 - 09/11/2026 06:00 PM - Greg Shah

Gap Details Storage

Purpose

A support level says **how much** of a 4GL feature FWD supports. It cannot say **what** is missing, which is the part a gap analysis actually has to write down. This change adds a place to record that text, carries it from the gap marking rules through to the Analytics reports, and displays and exports it alongside the level it explains.

It also corrects how a report category arrives at its level in the first place, which turned out to be necessary before details could mean anything.

In 11747a revision 16742, the gap storage solution has been committed.

Two details, not one

Conversion support and runtime support are independent, and a feature is usually limited on only one of them. So there are two texts per category,

cvt_gaps_detail and rt_gaps_detail, each explaining the level next to it. Either may be absent; most markings have no details at all, and that stays the normal case.

Detail text on a level of FULL is dropped with a warning. Full support means nothing is missing, so text there is almost always a leftover note rather than a gap.

The marking rules

A gap marking map entry is normally just a support level:

```
<rule>funcs.put(prog.kw_entry, rw.cvt_lvl_full | rw.rt_lvl_full)</rule>
```

Where there is something worth recording, the entry is written with rw.gap() instead:

```
<rule>
   queryOpts.put(prog.kw_max_rows,
                 rw.gap(rw.cvt_lvl_none | rw.rt_lvl_none,
                        "the option is discarded at conversion, with a warning",
                        "the query returns every row, the limit is not applied"))
</rule>
```

Both forms may appear in the same map. The marking rules read either one through rw.gapLevel(), rw.gapCvtGapsDetail() and rw.gapRtGapsDetail(), so the roughly 2,700 existing entries which have nothing to say were left exactly as they are. Only entries that gain details change shape.

Pass null for a side with no details, as in this entry where only conversion is limited:

```
<rule>
   funcs.put(prog.kw_can_find,
             rw.gap(rw.cvt_lvl_partial | rw.rt_lvl_full,
                    "nested CAN-FIND inside a WHERE clause converts to a client-side expression",
                    null))
</rule>
```

How the details travel

1. gaps/*.rules - the marking map value is either a level or a GapMarking.
2. gaps/gap_analysis_marking.xml - each of the 27 map lookups now keeps the raw value in gval and resolves the level from it. The details are read once, where the level is persisted, rather than at every lookup.
3. The AST node carries support_level_cvt_gaps_detail and support_level_rt_gaps_detail notes next to the existing support_level, written only when there is something to write.
4. reports/consolidated_reports.xml - reads those notes next to the existing support level expression and passes them to rw.addMatchMultiplexed().
5. category.cvt_gaps_detail and category.rt_gaps_detail store them.
6. ReportApi selects them into SummaryRow, and report.js displays them.

Step 4 is the reason no report definition changed. The details are read once in the shared report walk rather than through a per-report expression, so **every** report which already has a support level automatically reports the details behind it. profile.rpt was not touched.

Lowest wins

A report category aggregates every occurrence of a feature, and those occurrences need not be marked identically. The same built-in can be fully supported in ordinary code and restricted inside a WHERE clause, which is exactly what the WHERE clause marking from #11863 introduced.

Until now addMatchCategory() wrote the level only when it created the category, and there was no update category anywhere. Whichever occurrence happened to be walked first decided the level for all of them. This contradicted the "Gap Analysis" chapter, which describes the column as the lowest level available.

The level is now reduced to the weakest of the occurrences, one section at a time:

- Within a section the constants are already ordered so that a numerically smaller flag is the weaker level, so the reduction is a minimum.
- LVL_UNKNOWN is zero and would win every comparison, so an unset section is treated as "nothing was marked here" and the other level is taken.
- The two sections reduce independently, since conversion and runtime are independent.
- Each section's gap details travel with the level that wins it, so the text on display always explains the level beside it.

SupportLevelHelper gained cvtLevel(), rtLevel(), weaker() and lowest() for this.

Display and export

Two columns were added to the summary grid, Conversion Gaps after the Conversion level and Runtime Gaps after the Runtime level. They appear only on reports which have support levels, like the level columns themselves.

Gap details are written for a reader, not for a grid, and some run to a couple of sentences. On screen the cell shows the first 60 characters with an ellipsis, so a long explanation cannot make every row on the page as tall as itself. The full text is still reachable two ways:

- the cell tooltip shows it in full;
- the CSV download carries it in full, through an explicit accessorDownload on both columns. The formatter only shortens what is drawn; the download deliberately bypasses it, because exporting a truncated explanation would defeat the point of recording it.

Files changed

File	Change
src/com/goldencode/p2j/report/GapMarking.java	new; a support level paired with its two detail texts, plus static readers which accept either form of map value
src/com/goldencode/p2j/report/SupportLevelHelper.java	section isolation and the weakest-level combination
src/com/goldencode/p2j/report/ReportWorker.java	the rw.gap() family, the two new category columns, and the lowest-wins merge in addMatchCategory()
src/com/goldencode/p2j/report/ReportDefinition.java	cats2Gap, the per-category cache of the level and details currently stored
src/com/goldencode/p2j/report/server/ReportApi.java	select the two columns into the summary
src/com/goldencode/p2j/report/server/SummaryRow.java	the two new fields
src/com/goldencode/p2j/report/web/res/report.js	the two new columns, the abbreviation helper and the download accessors
rules/gaps/gap_analysis_marking.xml	keep the raw map value, resolve the level from it, persist the details
rules/reports/consolidated_reports.xml	read the notes and pass them through
rules/gaps/database.rules, rules/gaps/expressions.rules	four entries converted to rw.gap() as the first real details

Caveats

- The display truncation and CSV export were not exercised in a browser during this work. The mechanism is standard and the download accessor is explicit, but a visual check of the two new columns and one CSV download is worth doing before this is relied on.
- Only four entries currently carry details. The bulk migration of the existing end-of-line comments is separate work; of 2,764 marking entries, 1,031 carry a comment today, but 373 of those sit on fully supported levels and are explanatory notes rather than gaps, leaving about 658 candidates.
- A category created without ever passing through a level has no entry in cats2Gap and is left alone by the merge, which preserves the old behaviour for those.