Conversion Tools - Feature #11861

generate a complete gap analysis document for a project

09/10/2026 02:10 PM - Greg Shah

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	100%
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 02:25 PM - Greg Shah

The idea here is to add a facility to analytics which generates a complete gap analysis for an entire project as a single document.

Currently, I manually have to traverse reports (all of the ones that have support levels enabled) in the web UI to see each category that is marked with any support that is not CVT_LVL_FULL or RT_LVL_FULL. It is time consuming (and error prone) to type up each entry that is a gap.

Initially, if we just had a list of items with their non-full support levels, it would be enough to save a lot of time.

Each item might look something like this:

<keyword_or_category_descriptor> <4gl_feature_type> <cvt_support_level> <cvt_gap_details> <rt_support_level> <rt_gap_details>

It would also be useful to summarize the usage pattern from the associated details report. I can see 2 patterns here:

1. If there is a small number of usage locations (less than 11), then an explicit list of those locations can be provided.
2. If there are many usages (11 or more) then some summary like "X matches in Y files".

This would optimally be something generated by the ReportServer, driven by a request in the web UI. The output probably is best as formatted PDF. That can be converted in downstream usage to other forms (e.g. Textile).

#3 - 09/11/2026 06:50 AM - Alexandru Lungu

The raw topics generated in #11415 can be simply parsed by a script and a list (CSV, textile, etc.) can be generated. Example generate now from the **public** kb is below.

FWD 4GL Feature Gap Analysis - Conversion / Runtime Support

Keyword / Category	4GL Feature Type	CVT Support	CVT Gap Details	RT Support	RT Gap Details
ACTIVE-WINDOW	handle	Full	-	Full	-
AVAILABLE	attribute	Full	-	Partial	• **functional/sure:** Reading AVAILABLE through a BUFFER-FIELD handle only ever reports two of its three documented outcomes: the unknown value when the owning buffer holds no current record, and TRUE otherwise. The third outcome - the owning buffer holds a record but the referenced field is absent from it - is never produced by the current implementation, so a program that branches on a FALSE result from a buffer-field handle's AVAILABLE will not observe it. • **functional/sure:** Assigning the unknown value ? to AVAILABLE always raises error 4083 as a hard error (ERROR-STATUS:ERROR set TRUE), independent of the session's configured PROVERSION compatibility level. The reference implementation's own severity for this same assignment is version-gated - a warning below compatibility level 12.8, an error at 12.8 and above. A program written against a pre-12.8 compatibility level that relies on this assignment producing only a warning (ERROR-STATUS:ERROR left FALSE) will instead see an error raised. The sibling typed-value rejection (error 4052) carries no such divergence: it is always a warning on both implementations.
BLOB	data type	Partial	• **functional/sure:** The reference implementation rejects a large object in a WHERE clause at compile time. Here such a query is	Partial	• **functional/sure:** The reference implementation rejects a large object in a WHERE clause at compile time. Here such a query is

			accepted and executed. What it compares depends on the target database, so the query may silently return results that do not correspond to comparing the objects' content. • **functional/sure:** The large-object size options recorded against a BLOB field in the schema definition are accepted and then ignored. No size constraint from the source schema reaches the target database column, so a field declared with a modest maximum size accepts far larger values. • **functional/sure:** An initial value declared on a BLOB field is discarded without a diagnostic; the field is unknown when the record is created. A schema read from an external schema definition reports an error for the same construct instead. • **functional/sure:** A statically declared temp-table that carries a BLOB field but is not declared NO-UNDO converts successfully with a warning, where the reference implementation reports an error. The converted application then has an undoable large-object field whose undo behavior is undefined. • **functional/sure:** Defining a variable or parameter LIKE a BLOB field is not a supported construct, but instead of a diagnostic identifying the offending declaration, conversion of the file aborts with an internal failure. • **functional/sure:** A BLOB field is stored differently on each supported target database, with different capacity ceilings, and on one of them large objects are held outside the row and read through a mechanism that does		accepted and executed. What it compares depends on the target database, so the query may silently return results that do not correspond to comparing the objects' content. • **functional/sure:** Because a BLOB is not rejected as an index component, a schema that indexes a BLOB field can be converted. Depending on the target database the resulting index either fails outright when the index key is computed, or silently counts the field's full declared width against that database's index-key size limit. • **functional/sure:** No maximum size is enforced on a BLOB value at run time. The effective ceiling is whatever the host process and the target database allow, rather than a defined limit, and exceeding it surfaces as a memory or storage failure rather than a numbered error. • **functional/sure:** An initial value declared on a BLOB field is discarded without a diagnostic; the field is unknown when the record is created. A schema read from an external schema definition reports an error for the same construct instead. • **functional/sure:** Enabling undo on a temp-table that carries a BLOB field is reported at a different point and with a different error number than in the reference implementation, which reports it as the undo attribute is set. Error-handling logic that tests for a specific error number on that statement therefore behaves differently. • **functional/sure:** A BLOB field is stored differently on each supported target database, with different capacity ceilings, and on one of them large objects are

			not support modification in place. Maximum-size behavior, transactional behavior when reading a BLOB, and the effect of very large values therefore vary by target database rather than being uniform. • **performance/sure:** Loading data into a table that has a BLOB or CLOB column cannot use the fast bulk-load path and falls back to row-by-row insertion, making the initial data migration of large-object tables materially slower. • **functional/unsure:** Importing legacy data that contains BLOB fields requires the large-object export files to be placed in a dedicated subdirectory of the data export directory. A dump left in its original layout imports the affected fields as unknown rather than reporting a missing-file error. (#3500) • **functional/unsure:** Converting a BLOB to character data is rejected with a datatype error on some paths and silently succeeds on others, decoding the bytes with an unspecified encoding. Whether a given program hits the erroring path or the succeeding one depends on how the conversion is reached. • **functional/unsure:** The recognition attributes recorded for the BLOB type keyword - whether it may be abbreviated, and whether it is reserved - are carried from a bulk keyword import and marked unverified, so an unusual spelling, or the use of the word as an identifier, could be classified differently than in the reference implementation.		held outside the row and read through a mechanism that does not support modification in place. Maximum-size behavior, transactional behavior when reading a BLOB, and the effect of very large values therefore vary by target database rather than being uniform. • **functional/unsure:** On one supported target database configuration, storing or retrieving a BLOB larger than roughly 20 KB can fail with a storage-level error rather than round-tripping the value. (#4421) • **functional/unsure:** Reading and writing large objects is not uniform across the supported target databases, and temporary tables behave differently again from permanent tables on the same target. The same program can therefore succeed against one target database and fail against another. (#4421) • **performance/sure:** Retrieving a record from a table that has a BLOB field brings the whole large-object value into memory even when the program never references that field. Scanning such a table therefore costs in proportion to the total large-object volume rather than to the row count, and the value is also materialised in full before rows are sent to a remote client. (#3500) • **functional/sure:** Copying a record that carries a BLOB field can leave the two copies sharing the same large-object content rather than each holding an independent value, so a later change made through one is visible through the other. Setting a BLOB field unknown discards its content immediately,

					with no prior state to restore. • **functional/sure:** In a dynamic context where the value's type is not known until run time, offering a non-binary value where a BLOB is expected can leave the BLOB holding an empty value instead of raising the datatype error that the equivalent statically typed assignment raises. • **functional/unsure:** Converting a BLOB to character data is rejected with a datatype error on some paths and silently succeeds on others, decoding the bytes with an unspecified encoding. Whether a given program hits the erroring path or the succeeding one depends on how the conversion is reached. • **functional/sure:** Some copy failures into a BLOB target that the reference implementation reports as errors complete silently instead, notably an overlay write positioned past the end of the target. Code that relies on the error condition to detect a failed copy sees a successful one. (#6457) • **functional/unsure:** Requesting the export-format representation of a BLOB value yields nothing, where the equivalent request on a CLOB or a RAW value yields a representation. The main export path is unaffected, so this is only observable through constructs that format a value in export form without writing it to a stream. • **functional/unsure:** A multi-byte write into a BLOB completes without changing the value and without reporting an error, while the corresponding single-byte write does

					take effect.
BROWSE	widget	Partial	• BROWSE is broadly functional across all three drivers but each driver carries documented gaps. The abstract base class itself records open questions about whether NO-ASSIGN, NO-VALIDATE, and VALIDATE functions are fully honored.	Partial	• BROWSE is broadly functional across all three drivers but each driver carries documented gaps. The abstract base class itself records open questions about whether NO-ASSIGN, NO-VALIDATE, and VALIDATE functions are fully honored.
BUFFER	handle	Full	-	Partial	• **functional/unsure:** A DEFINE BUFFER whose scope is restricted to an iterative block (a FOR EACH, DO WHILE, DO ... TO, or REPEAT body) may not undergo a per-iteration release cycle. The reference behavior clears the buffer's current-record cursor and downgrades or releases its locks at the top of each iteration; observed behavior can leave the current record and lock state carried over across iterations. Code that expects each iteration to see the buffer with AVAILABLE reporting FALSE before a fresh FIND may instead observe a stale current record from the previous iteration. (#11236) • **functional/sure:** The IS-PARTITIONED attribute reports FALSE for every buffer at runtime, regardless of whether the buffer's underlying table is actually a partitioned table. Applications that branch on this attribute to enable partition-aware code paths or to inspect a table's storage layout will not observe the true partitioning status of the underlying table.
BUFFER-CREATE	method	Full	-	Full	-
CAN-FIND	function	Full	-	Full	-
CHAR	data type	Full	-	Full	-
CHARACTER	data type	Partial	• **functional/sure:** When a value declared CASE-SENSITIVE is compared against a	Partial	• **functional/sure:** When a value declared CASE-SENSITIVE is compared against a

			text field that was not declared case-sensitive - for example in the WHERE clause of a FIND or a FOR EACH - the comparison matches here, where the reference implementation reports no match. The reference implementation does not match that pairing at all, not even for values identical character for character, and likewise does not match when the two differ only by a trailing space or share a leading space. The opposite pairing diverges the other way: under some query-caching configurations, a case-sensitive field compared against a value that is not case-sensitive can report no match where the reference implementation matches. (#7352, #7108) • **functional/unsure:** When both sides of a comparison are text written directly in the program, the comparison is performed without removing trailing spaces and without regard to a case-sensitive context. Two literals differing only by trailing spaces therefore compare unequal, where the same comparison between a variable and a literal compares them equal. • **functional/sure:** How much a CHARACTER field can hold depends on which database the application was converted for. On some targets the column is unbounded; on others its width is derived from the field's declared maximum width, or from twice the width of its display format, falling back to sixteen characters when the format cannot be read. An		text field that was not declared case-sensitive - for example in the WHERE clause of a FIND or a FOR EACH - the comparison matches here, where the reference implementation reports no match. The reference implementation does not match that pairing at all, not even for values identical character for character, and likewise does not match when the two differ only by a trailing space or share a leading space. The opposite pairing diverges the other way: under some query-caching configurations, a case-sensitive field compared against a value that is not case-sensitive can report no match where the reference implementation matches. (#7352, #7108) • **functional/sure:** Converting text between code pages is incomplete. Obtaining a character code for a character, and converting a code back to a character, with an explicit source and target code page, give results that differ from the reference implementation for several common code pages. A code obtained from a value longer than one character cannot be converted back to that value. Redirecting output with an explicit code-page conversion, and fixing a value's code page, are affected by the same incompleteness, and characters with no representation in the target code page are the least reliable case. Separately, a request to convert a text value to another code page returns the value unchanged and only re-labels it, so a later byte-oriented operation sees the

			application converted for one target can therefore reject a value that the same application converted for another target accepts, and the width is fixed when the schema is converted rather than when the application is deployed. • **functional/sure:** On one supported target an indexed CHARACTER column whose derived width exceeds four thousand characters is silently narrowed to that width when the schema is converted, with only a log message. Storing a longer value then fails with an error reporting that the value is too large for the type - a condition the reference implementation does not have, because a CHARACTER field there carries no declared column width. • **functional/unsure:** The storage width of a CHARACTER field of a temp table is not derived from the field's declaration. On targets that size text columns, such a field gets the fallback width rather than the width its display format implies, so a value that fits in the same field of a permanent table can fail to be stored in the temp table. • **functional/unsure:** For a CHARACTER field declared with an extent, each element's column is given the whole field's declared maximum width rather than a per-element share of it. The columns are wider than needed, which loses no data but does count against the index key size limit of targets that have one. • **functional/sure:** A wide CHARACTER field can be made part of an index whose key exceeds the target database's key size		original bytes. (#9532, #4766, #2167) • **functional/sure:** The default internal code page is a multi-byte one, where the reference implementation's default is single-byte. Byte-oriented operations on text therefore give different results unless the internal code page is set explicitly at startup: the byte length of a value holding characters outside the ASCII range is larger, and a byte-oriented slice of such a value cuts at a different point. • **functional/sure:** The facilities for handling multi-byte text are incomplete. A request for the display-column width of a value returns its character count instead, so output aligned from that number is misaligned for double-width characters, and no error is reported. A test for whether a byte is the leading byte of a multi-byte character yields the unknown value, so neither branch of such a test is taken. • **functional/sure:** Case-insensitive comparison and the case-changing built-ins fold letters using the locale of the machine the runtime is running on, rather than the case table of the session code page. For letters whose case mapping is locale-dependent, two values that should compare equal case-insensitively may not, and the outcome can change with the server's configuration. • **functional/sure:** Ordering of text values follows code point order after case folding. A collation is not applied, so values order differently from a collation-driven ordering wherever the two disagree, notably for accented letters and for punctuation. A

			limit. Only one supported target checks for this, it checks only the unique indexes, and it reports the problem as a warning rather than refusing the index, so the failure can surface later as a database error when the schema is applied. • **functional/sure:** A display format on a CHARACTER declaration that the runtime cannot read is discarded when the application is converted, with no diagnostic. The field is then laid out with the default width instead of the intended one, so a value may appear truncated or padded differently. • **functional/sure:** The text-matching operators require a CHARACTER left operand inside a query. A LONGCHAR left operand fails conversion with an incompatible-datatypes error, although the same operator accepts a LONGCHAR outside a query. • **functional/sure:** When the type of an expression cannot be determined while the application is converted, the expression is formatted as though it were CHARACTER and a diagnostic is printed. A numeric or date expression in that position is displayed with a text format rather than with its own. • **functional/unsure:** CASE-SENSITIVE given as an option on a frame field is dropped rather than applied, so the field displays and compares with the default sensitivity. • **functional/unsure:** When a CHARACTER value is rendered back into the source form of a text literal, a non-printable character whose code is above 255 is dropped from the		collation name supplied to the comparison built-in is accepted and has no effect. • **functional/unsure:** The comparison built-in implements only the case-sensitive and case-insensitive strengths. A comparison requesting one of the collation-level strengths yields the unknown value rather than a result, and the raw strength is treated as identical to case-sensitive, so a comparison of multi-byte text that the reference implementation distinguishes between the two agrees with only one of them. A comparison requesting the pattern-matching operator also yields the unknown value. • **functional/sure:** A request to normalise a text value returns it unchanged. Values that differ only in how an accented character is composed therefore continue to compare unequal, where normalising them first would make them equal. • **functional/sure:** The maximum length of a CHARACTER is reported only when a longer LONGCHAR value is assigned into one. A value grown past the same ceiling by concatenation, by filling, or by an in-place replacement is accepted with no error, so the limit is advisory on those paths and an over-long value can be carried into parts of a program that assume the limit holds. • **functional/sure:** The byte length at which the maximum-length error is reported is measured with the encoding of the environment the runtime is running in, rather than with the

			result rather than escaped, so the rendered literal is shorter than the value it came from and does not reproduce it. • **functional/sure:** A CHARACTER field may declare a per-field code page in its schema definition, the same option a CLOB field accepts. The built-ins that read a field's code page back behave differently depending on how the call is written: naming the CHARACTER field directly is rejected at conversion as a type mismatch, while reaching the same field indirectly through a buffer-field handle is accepted syntactically but then fails at runtime with an incompatible-datatypes error. Neither path returns a code page for a CHARACTER field. (#6442) • **functional/unsure:** The one, two and three letter abbreviations of the type name are accepted in a variable, parameter, class-property and field declaration, but not where the type designator stands alone as a function or method return type; there, only the four-letter form CHAR and longer spellings are recognised.		session code page. The same program can therefore accept an assignment on one deployment and reject it on another, and the threshold does not correspond to the limit the runtime's own diagnostics state. • **functional/sure:** A parameter of a dynamic call declared CHARACTER accepts only a CHARACTER value; supplying any other type, a LONGCHAR included, raises an incompatible-datatypes error, although the same conversion succeeds silently in an ordinary assignment. The relation is not symmetric: a parameter declared LONGCHAR does accept a CHARACTER value. • **functional/unsure:** When both sides of a comparison are text written directly in the program, the comparison is performed without removing trailing spaces and without regard to a case-sensitive context. Two literals differing only by trailing spaces therefore compare unequal, where the same comparison between a variable and a literal compares them equal. • **functional/unsure:** A CHARACTER value can come to hold an embedded null character, and the handling of such a value is not a faithful reproduction of the reference implementation. On the ordinary path the null character and everything after it are replaced with spaces, preserving the value's length; on other paths the value is cut at the null. Operations on a value holding a null character may therefore differ from the reference implementation. (#2429) • **functional/unsure:**

					A CHARACTER field of a temp table defined from a permanent table's definition does not reliably take that table's code page. A value stored into and read back from such a field may be encoded differently from the same value in the permanent table the definition came from. (#4520) • **functional/sure:** How much a CHARACTER field can hold depends on which database the application was converted for. On some targets the column is unbounded; on others its width is derived from the field's declared maximum width, or from twice the width of its display format, falling back to sixteen characters when the format cannot be read. An application converted for one target can therefore reject a value that the same application converted for another target accepts, and the width is fixed when the schema is converted rather than when the application is deployed. • **functional/sure:** On one supported target an indexed CHARACTER column whose derived width exceeds four thousand characters is silently narrowed to that width when the schema is converted, with only a log message. Storing a longer value then fails with an error reporting that the value is too large for the type - a condition the reference implementation does not have, because a CHARACTER field there carries no declared column width. • **functional/sure:** A wide CHARACTER field can be made part of an index whose key exceeds the target database's key size

					limit. Only one supported target checks for this, it checks only the unique indexes, and it reports the problem as a warning rather than refusing the index, so the failure can surface later as a database error when the schema is applied. • **functional/unsure:** When the conversion of a CHARACTER parameter fails, the accompanying secondary error reporting that the caller's output parameter could not be converted is not raised. A program that counts the pending error messages, or reads them by position, therefore sees a shorter list than the reference implementation produces. • **functional/sure:** A CHARACTER field may declare a per-field code page in its schema definition, the same option a CLOB field accepts. The built-ins that read a field's code page back behave differently depending on how the call is written: naming the CHARACTER field directly is rejected at conversion as a type mismatch, while reaching the same field indirectly through a buffer-field handle is accepted syntactically but then fails at runtime with an incompatible-datatypes error. Neither path returns a code page for a CHARACTER field. (#6442)
CHARACTER	literal	Partial	• **functional/sure:** A quoted literal whose last character before the closing delimiter is a tilde is accepted by the reference implementation inside a schema definition file, but is not terminated here: the closing delimiter is consumed as an escaped character, the literal runs on past its intended end, and the surrounding	Partial	• **functional/sure:** A literal written inside a query or expression string that is assembled and parsed while the program runs is not decoded by the same rules as a literal written directly in source: an embedded linefeed is dropped, and an octal escape must be spelled with all three digits. The same literal text can

			definition fails to import. (#5399) • **functional/sure:** Hexadecimal (~uXXXX, ~UXXXXXX), octal (~nnn) and null-character escapes inside a literal do not always yield the value the reference implementation produces. A hexadecimal escape can survive as ordinary text instead of the character it denotes, a null escape can blank or truncate the remainder of the value at a different point, and a tab written inside a literal can be replaced by spaces that the reference implementation preserves. (#10028, #3971, #6308) • **functional/unsure:** An alternative-coding escape sequence written immediately after a closing delimiter, with no whitespace in between, is treated specially by the reference implementation for some sequences and not others. Most of those forms are reproduced, but at least one still yields different text than the reference implementation. (#6308, #6859) • **functional/sure:** Whether a backslash inside a literal acts as a second escape character or as ordinary content is chosen once for an entire project rather than per source file. A code base whose sources assume both conventions cannot have both interpreted correctly in a single conversion. (#2366) • **functional/sure:** Justification and size options are applied to a literal standing on its own, but are ignored when the same literal supplies a widget label, a title, help text or a display format. In those positions the		therefore produce two different values depending on where it is written.

			value appears without the requested padding or truncation. (#10290) • **functional/sure:** When a literal carries a size option and also contains escape sequences, the size is measured against the escape text rather than against the characters the escapes denote. The value is padded to the wrong width, and a truncation can cut an escape sequence in half. • **functional/unsure:** A size option written on its own, with neither a justification letter nor a translatability flag (for example "abc":20), is accepted but leaves the value unchanged instead of padding or truncating it to the requested width. Adding either one - "abc":L20 or "abc":U20 - makes the same size take effect. (#10290) • **functional/unsure:** The undocumented single-letter flag that may appear wherever the untranslatable flag may appear, but never alongside it, is accepted and, like the untranslatable flag, is enough on its own to bring a size option into effect, so "text":x8 is padded to eight characters. What it does not do is exclude the literal from translation. Whether it carries a distinct meaning in the reference implementation, or should bring a size into effect there at all, is unconfirmed. (#1514, #10290) • **functional/unsure:** Passing a single literal as the argument of a class method whose parameter is declared LONGCHAR can produce output that does not build, because the literal is carried through as plain text without being promoted to the parameter's declared type. The related case		

			of a concatenation of literals in the same position has been corrected. (#10973) • **functional/sure:** A literal written inside a query or expression string that is assembled and parsed while the program runs is not decoded by the same rules as a literal written directly in source: an embedded linefeed is dropped, and an octal escape must be spelled with all three digits. The same literal text can therefore produce two different values depending on where it is written.		
COPY-LOB	statement	Full	-	Partial	• **functional/unsure:** For at least one character outside the common character range, copying a character value through COPY-LOB with an explicit target codepage may produce an output byte sequence of a different length than the reference implementation for the same character and target codepage. (#6457) • **functional/sure:** When a COPY-LOB source is both shorter than the requested offset/length and contains a character invalid for the target codepage, the error raised for this combined condition differs from the error the reference implementation raises first for the same case, so NO-ERROR / error-handling code can observe a different error number. (#6457, #4768) • **functional/unsure:** COPY-LOB against a source file path that does not exist may raise a single error where the reference implementation raises two chained errors (a file-not-found error followed by a COPY-LOB-specific copy-failure error) for the same condition. (#6457) • **functional/unsure:**

					Byte-order-mark handling for COPY-LOB file targets written under multi-byte target codepages has not been fully verified against the reference implementation, so byte-order-mark fidelity for those targets is unconfirmed rather than confirmed either way. (#6457) • **performance/unsure:** Very large character (LONGCHAR/CLOB) values copied through COPY-LOB are still fully materialized in memory during the operation; only binary (MEMPTR/BLOB) copies use a bounded-memory streaming transfer, so an extremely large character copy may have different memory characteristics than a binary copy of comparable size. (#11327)
DBPARAM	function	Full	-	Partial	• **functional/sure:** The single-user-mode connection flag is accepted at connect time and reported back by DBPARAM, but it has no enforcing effect: the database is not actually restricted to one client as a result of supplying it. (#3813) • **functional/sure:** Only a fixed, enumerated set of database connection options is recognized. Supplying an option outside that set does not raise an error or warning: the CONNECT statement completes as if it had succeeded, but the database is left unconnected - so DBPARAM (and every other operation on that logical name) behaves as if the CONNECT had never been issued, and the only way to notice is checking the database's connected state afterward. (#3813) • **functional/unsure:** For a database connected without an explicit CONNECT

					option list (for example an implicitly or automatically connected database), the parameter string DBPARAM returns may cover only a narrower, fixed subset of fields, rather than the fuller set reported for a database connected with explicit CONNECT options. • **functional/unsure:** The numeric-index form numbers currently connected databases in the order they were connected. Whether this ordering always matches the reference implementation's own numbering after databases are connected, disconnected, and reconnected in non-trivial sequences has not been independently confirmed.
DBRESTRICTIONS	function	Full	-	Partial	• **functional/sure:** The optional table-name argument has no effect on the result. Whether it is supplied, omitted, or names a table that does not exist, the function reports the same database-level restriction status. • **functional/unsure:** Only a read-only-connection restriction is ever reported: the function checks for exactly one connect-time restriction category out of a much larger set of connect-time options the runtime recognizes. Whether the reference implementation reports any additional restriction category that this function does not check for at all remains an open question rather than a confirmed non-issue. (#6454)
DECIMAL	literal	Full	-	Full	-
END-KEY	event	Full	-	Partial	• **functional/unsure:** When a block or frame that has an ON ENDKEY exit-condition handler is exited by the end-key gesture while

					a widget-level END-KEY trigger is also registered for that gesture (directly, or on a containing window or frame), the end-key exit condition can fail to be raised at all, or the wrong handler in the containment chain can be matched, so the block does not unwind the way the reference specification calls for. The reference specification always raises the end-key exit condition once the trigger has run; the observed divergence is in how far up the window/frame/widget chain the runtime searches for the matching handler, and in whether the last-pressed-key state reflects the end-key-triggering key at the point the condition is evaluated. (#6797) • **functional/sure:** Pressing the end-key gesture while an interactive field update is in progress correctly ends the block through the end-key condition, but the values that SCREEN-VALUE and the INPUT function report afterward for the fields in that frame do not reflect what was actually on screen at the moment the gesture was pressed. Reported values can differ from both the in-progress edited value and the field's pre-edit value. (#2527)
FIND-FIRST	statement	Full	-	Full	-
INT	data type	Full	-	Full	-
LENGTH	function	Partial	• **functional/unsure:** The converter previously wrapped LENGTH(r, "RAW") inside a DO ... TO loop bound with a wrapper whose declared return type disagreed with the underlying runtime entry point's int64 return, so the generated Java for the loop bound did not compile. (#11149)	Partial	• **functional/sure:** For LENGTH(<text>, "COLUMN") the runtime's per-code-point width table differs from the reference implementation's empirical per-code-point mapping for a subset of Unicode code points. Both agree on ASCII and standard CJK glyphs; they

			• **functional/sure:** LENGTH called with the literal unknown value as its first argument (LENGTH(?)) is rejected during conversion with numbered error 223 ("Incompatible data types in expression or assignment"). The reference implementation accepts this call and returns the unknown value at runtime, so source that compiles and runs under the reference cannot be converted when the first argument is a bare ? literal. (#11149) • **functional/unsure:** LENGTH called with an array-typed value as its first argument - for example, a whole-EXTENT value returned by a dynamically invoked method - may not surface the reference implementation's numbered error 12117 ("Unacceptable datatype for LENGTH argument"). Code that relies on the numbered error to trap this invalid-input case may not observe the same error condition under FWD.		disagree on a small number of exotic code points, so LENGTH(ch, "COLUMN") for those code points returns a value that the reference implementation does not produce. (#6389) • **functional/sure:** When the unit argument to LENGTH is an expression evaluated at runtime (e.g. LENGTH(ch, TRIM("+++") + "+++")), the runtime dispatches through a different validation path in GUI mode than in ChUI mode. The GUI path raises error 1186 for expressions the ChUI path accepts as valid character unit specifiers, so the same call succeeds under ChUI and fails under GUI. (#6389) • **functional/unsure:** For character values encoded outside the session's current codepage (e.g. UTF-16-encoded strings when cpinternal is ISO8859-1), the runtime returns 0 uniformly for character, raw, and column unit measurements. The reference implementation's behavior for the same codepage-mismatch scenario is contested across the ticket's journals. (#6389) • **functional/unsure:** LENGTH called with an array-typed value as its first argument - for example, a whole-EXTENT value returned by a dynamically invoked method - may not surface the reference implementation's numbered error 12117 ("Unacceptable datatype for LENGTH argument"). Code that relies on the numbered error to trap this invalid-input case may not observe the same error condition under FWD.
LOGICAL	data type	Full	-	Full	-

MAXIMUM	function	Full	-	Partial	• **functional/unsure:** When MAXIMUM (or MINIMUM) appears inside a dynamic query - one built through a query handle and prepared with a query string - the query can fail at open time. The first execution after server startup succeeds, but subsequent executions of the same query raise a runtime error instead of returning the computed maximum. The number and types of arguments do not change the behavior, and the same expression evaluated outside a dynamic query is unaffected. As a workaround, compute the MAXIMUM into a variable and reference that variable in the query string. (#8952)
MINIMUM	function	Full	-	Partial	• **functional/unsure:** When MINIMUM (or MAXIMUM) appears inside a dynamic query - one built through a query handle and prepared with a query string - the query can fail at open time. The first execution after server startup succeeds, but subsequent executions of the same query raise a runtime error instead of returning the computed minimum. The number and types of arguments do not change the behavior, and the same expression evaluated outside a dynamic query is unaffected. Ordinary functions such as ABSOLUTE and SQRT are not affected. As a workaround, compute the MINIMUM into a variable and reference that variable in the query string. (#8952)
PERIOD	punctuation	Full	-	Full	-
PLUS	operator	Full	-	Full	-
Progress.Lang.Object	class	Full	-	Full	-
REPOSITION-BACKWARDS	method	Full	-	Partial	• **functional/sure:** When the rows argument evaluates to the unknown value

					(?), REPOSITION-BACKWARD returns FALSE without raising or recording any numbered error; ERROR-STATUS is left unset. This departs from the usual convention elsewhere in the reposition family, where a FALSE return is normally paired with a structured error. • **functional/unsure:** It is unconfirmed whether a block configured to intercept errors (BLOCK-LEVEL ON ERROR UNDO, THROW) can catch and unwind on a REPOSITION-BACKWARD failure the way it can for the related REPOSITION-TO-ROWID method and the REPOSITION statement, both of which received a dedicated error-handling correction for exactly this class of problem. REPOSITION-BACKWARD was not named in that fix's scope, and current behavior for it specifically has not been separately confirmed. (#10277)
REPOSITION-FORWARDS	method	Full	-	Partial	• **functional/unsure:** When the rows argument evaluates to the unknown value (?), REPOSITION-FORWARD returns FALSE without raising or recording any numbered error; ERROR-STATUS is left unset. Whether this matches the reference implementation's contract for an unknown-value argument in this position is not corroborated by an independent source. • **functional/unsure:** On a dynamically created query that has no cursor associated with it (distinct from the closed and not-SCROLLING cases above), REPOSITION-FORWARD does not fail with

					one of the numbered errors listed above and is not suppressed by NO-ERROR the way those are; it fails as an unrecognized runtime condition instead. Whether this matches the reference implementation's error contract for the same situation is not corroborated by an independent source.
TRIGGER-PROCEDURE	statement	Partial	• **functional/sure:** Declaring the value-only ASSIGN form (naming only a new working variable, with no OF table.field reference) does not reliably produce a working trigger: the generated trigger's value typing and datatype checking are not complete for this specific form, so its behavior should not be relied on. The field-bound ASSIGN form (ASSIGN OF table.field) is not affected. (#2223) • **functional/sure:** When the OF clause names its target table using an abbreviation that is also used elsewhere in the same procedure to refer to that table's default buffer, the trigger's buffer and that default buffer are treated as one and the same, rather than as two independent buffers. This can cause unexpected sharing of buffer state between the trigger and the rest of the procedure when an abbreviated table reference is used. (#4492)	Partial	• **functional/sure:** Declaring the value-only ASSIGN form (naming only a new working variable, with no OF table.field reference) does not reliably produce a working trigger: the generated trigger's value typing and datatype checking are not complete for this specific form, so its behavior should not be relied on. The field-bound ASSIGN form (ASSIGN OF table.field) is not affected. (#2223) • **functional/sure:** In one confirmed scenario, when a FIND statement relocates to a record that is already resident in a buffer under a different cursor, a pending WRITE event on that buffer can fire later than expected - observed to fire only after a subsequent DELETE event on the same record, instead of before it. Other WRITE-timing scenarios match the reference implementation; this buffer-reuse case is the one confirmed residual difference. (#2222) • **functional/unsure:** In a multi-user scenario, when a record is found to be stale (already changed by another session) on a table with a WRITE trigger, closing out the enclosing transactional block can occasionally surface an unrelated-looking runtime error instead of the expected

					stale-record condition. This report is not yet confirmed as reliably reproducible. (#11552)
VALUE-CHANGED	event	Full	-	Full	-
WHERE	phrase	Full	-	Full	-

#4 - 09/13/2026 05:43 PM - Greg Shah

Branch 11747a revision 16746 has the implementation of the gap analysis documentation generation tool.

#5 - 09/13/2026 05:46 PM - Greg Shah

Project Gap Analysis Document

Introduction

FWD Analytics already answers "what does this application use?" in detail: a hundred and more reports, each listing the categories of one 4GL feature with a support level attached. What it has not answered is the question a project actually starts with, which is "what in this application does FWD not yet support, and how much of a problem is that?"

This describes the facility which answers it. It reads the analytics report database and writes one document, organised the way a reader needs it rather than the way the reports are structured: every 4GL feature the project uses which FWD does not support completely, filed under the part of the language it belongs to, worst first, with what is missing and how much the project depends on it.

This was built on top of the gap detail storage of #11860. Nothing here changes how support levels are decided; it changes what is recorded alongside them and how the result is read.

Design

What is an item

An item is one 4GL feature, not one report row. That distinction is the whole of the design, because the reports overlap heavily: CAN-FIND is a category in *Builtin Function Usage*, again in *Database Builtin Functions/Variables* and again in *Built-In Functions Called in WHERE Clauses*. Three rows, one gap.

Every item carries a stable identity, built from three parts:

```
group / feature type / normalized name
base-language / function / can-find
```

The identity is stable across runs and across projects, so the same feature is the same item wherever it turns up. That is what makes it possible to merge the duplicates, and it is what a future annotation mechanism would key on.

The three groups

Every item is filed under exactly one of three top level groups, in this reading order:

1. **Base Language** — the language itself: statements, functions, expressions, error handling, sessions, security, codepages, and the built-in OO class library.
2. **Database** — buffers, queries, temp-tables, field handles, metaschema, embedded SQL.
3. **User Interface** — widgets, frames, windows, the clipboard, and the Windows-only facilities.

The important decision here is that **a group describes the 4GL feature, not the report it appears in**. A single report routinely spans all three: *Builtin Function Usage* holds ROWID (database), FRAME-COL (user interface) and SUBSTRING (base language). Filing by report would put all three in one place and be wrong for two of them.

So the group is recorded by the gap marking entry, which is where the feature is named in the first place. Of the twenty-four lookup maps in gap_analysis_marking.xml, nineteen hold entries of a single group and declare it once for the whole map. Five genuinely mix groups — attrmeth,

stmts, funcs, globalVars and builtinClasses — and name one per entry.

What makes a report a source of items

A new per-report attribute, featureType, names the kind of 4GL thing a report's categories are: statement, function, attribute, data type, block option and so on. Its presence is what makes a report contribute to the document at all.

That is deliberate. Many reports list an application's own names — field names, variable names, class names, procedure names, RUN targets — and an application name is never a gap. The 408 field-name rows across three reports in one customer project are all blob and clob, which appear as exactly two rows in *Database Field Usage (By Data Type)*. Those reports carry no featureType and contribute nothing.

Of the reports which carry support levels, sixty-four have a featureType and forty-one deliberately do not.

Presence is a gap

COM automation, ActiveX controls, window handles and .NET are not partially supported features. They reach outside the 4GL to a Windows facility a converted application does not have, so there is no support level to report and none of those reports carries one. Any use at all is the gap.

These reports are told apart by carrying a featureType but no support level expression, and they produce their own section of the document rather than support level tables.

Duplicates

Two reports seeing one feature are merged, in two passes.

The first pass merges by identity. Two candidates with the same group, feature type and name are the same feature by construction. A built-in class turns up in *Class Name References* wherever it is named and in *OO References* wherever it is used, and the first set is part of the second; reporting it twice with different counts would say nothing except that the two reports ask different questions. The candidate which saw the most occurrences is kept.

The second pass merges by occurrence set, which is what catches the same feature under different names. A name comparison misses KW_CAN_FIND against can-find() entirely; what is reliable is that both matched the same file, line and column every time. A narrower occurrence set is a different, narrower gap and stays its own item.

Comparing occurrence sets for every pair would mean reading every match in the project, so candidates are bucketed on a cheap fingerprint first and only collisions are confirmed exactly.

On a merge the more specific report wins the support level and the citation, but not necessarily the name: some reports name a feature by the parser token behind it, and a reader of the analysis knows the 4GL rather than FWD's token names. So can-find() wins over KW_CAN_FIND while the citation stays with the more specific report.

One keyword, several use cases

A marking entry is keyed by the 4GL keyword alone, with no receiver. attrmeth.put(prog.kw_clear, ...) is one entry covering every use of CLEAR, and CLEAR is four unrelated features: it empties a temp-table or a dataset, resets a CALL object, and clears what a frame or browse displays.

Splitting the marking is not the answer, and cannot be done reliably in any case: the receiver of an access through a plain HANDLE variable is only known at runtime. Instead the item keeps one group and the document prints a **summary of the usages with examples** underneath it — the kinds of thing the feature is used on, how many uses each accounts for, and a real line of source for each. The reviewer reads the split rather than inheriting a guess about it.

This is applied by rule, not by a list: the breakdown is computed for every attribute and method item, and printed whenever the item has more than one distinct receiver. Nothing has to maintain a register of which keywords are ambiguous, and a keyword which turns ambiguous in a later project is picked up on its own.

Unknown levels

An item whose support level is unknown is not a statement about the feature. It means the gap marking never reached it, which is a defect to fix rather than evidence of anything. Those items are always included and are called out as needing review, so that the marking gets corrected instead of quietly reporting a project as unsupported.

Implementation

Report database

Two columns were added.

Table	Column	Holds
report	featuretype	The kind of 4GL feature a report's categories name, or null if the report does not describe the 4GL
category	gap_group	The gap analysis group of the marked feature

Both are populated by the normal report run; no separate pass is involved.

Carrying the group

The group rides on the gap marking map value, the same way the gap details added by #11860 do. A map entry is either a plain support level or a GapMarking, and the marking rules read either form.

Function	Purpose
rw.setGapGroup(map, group)	Declare the group shared by every entry of a map. Applies it to each entry which does not already name one, so it can be combined with per-entry groups
rw.gap(group, lvl)	Name a group on one entry
rw.gap(group, lvl, cvt, rt)	Name a group and the gap details on one entry
rw.gapGroup(value)	Read the group back out of a map value
rw.gapGroupFor(map, qname)	Look a group up by qualified name against a map of name prefixes, longest prefix winning

The three groups are named by the constants rw.group_base_lang, rw.group_database and rw.group_ui, which resolve through ReportWorker.resolveConstant().

At marking time the group is written as a note, support_level_gap_group, alongside the support level itself. The report walk reads the note and passes it to addMatchCategory(), which stores it in category.gap_group. That is the same path the conversion and runtime gap details already take.

Where the groups are declared

The nineteen unambiguous maps declare their group once, at the end of the init-rules in gap_analysis_marking.xml:

```
<rule>rw.setGapGroup(typeMatch     , rw.group_base_lang)</rule>
<rule>rw.setGapGroup(fieldRefs     , rw.group_database)</rule>
<rule>rw.setGapGroup(uiEvents      , rw.group_ui)</rule>
```

The five mixed maps name a group on each entry that needs one:

```
<rule>funcs.put(prog.kw_can_find, rw.gap(rw.group_database,
                                         rw.cvt_lvl_partial | rw.rt_lvl_full,
                                         "nested CAN-FIND inside a WHERE clause converts to a client-side expres
sion",
                                         null))</rule>
<rule>attrs.put(prog.kw_search  , rw.gap(rw.group_ui, rw.cvt_lvl_full | rw.rt_lvl_full_restr))</rule>
```

Assignment is demand driven: an entry needs a group only once the feature actually surfaces as a gap. An entry without one defaults to base language, which is why the default is the group most features belong to.

A group written on an entry always beats the map's declaration, so a map can have a default and an exception at the same time. CONTAINS is the one such case today: it lives in typeMatch, whose declaration is base language, but it is the word index operator and therefore a database feature.

Built-in OO classes

builtinClasses is the one map which cannot hold a group next to a level, because it does not hold levels either. A built-in OO class is marked from the LegacyResourceSupport annotation on FWD's own implementation class, and the map is only a cache of what that lookup returned.

Those classes are grouped by package instead, in a twenty-fifth map, ooGroups, populated by addOoGroups() in gaps/expressions.rules. Everything defaults to base language, which is what the built-in class library mostly is — collections, strings, memptrs, JSON, reflection, HTTP, the web handlers. One package is not:

```
<rule>ooGroups.put("openedge.businesslogic.", rw.group_database)</rule>
```

rw.gapGroupFor() takes the longest matching prefix, so a class can still name its own group against the package it sits in.

The document model and renderer

Five classes in com.goldencode.p2j.report.server:

Class	Responsibility
GapItem	One item: group, feature type, name, level, gap details, counts, merged reports, usage breakdown
GapDocument	The document: items by group, the presence sections, the statistics, and the support level tiers
GapDocumentBuilder	Reads the report database and builds the model
TextileGapRenderer	Renders the model as Redmine Textile
GapDocumentDriver	Command line entry point

The model carries no presentation. Rendering to Textile is one renderer; anything else can be added without the analysis being redone.

Running inside a session

The build runs inside a report server session, which is what makes it honour the active file filter. A project's report database holds every file the conversion parsed, and the filter profile is what says which of them the project actually converts; reporting without it would report gaps in code the project does not convert.

The driver therefore opens the database the way the report web server does and establishes a session of its own, through two methods added to ReportApi: applyDefaultFileFilter(), which populates active_file for the session and names the profile applied, and releaseFileFilter(), which removes the session's rows afterwards. active_file is a global temporary table shared by every session and discriminated by a session id, so the driver uses a negative id, which the web server's own numbering can never produce.

Statistics

The statistics size the gaps rather than repeat them. A hundred features at partial support mean one thing in a twenty thousand line application and another in a two million line one.

Figure	Source
Source files by extension	The file table, counted by path within the filtered set
Lines of code, direct and included	The Lines of Code Analysis By File report
Include files and references	Include File Usage (By Included Filename)
Databases, tables, fields, indexes, sequences	The schema reports, by title
Possible gaps, and those needing review	The document itself

Lines of code are taken from the existing analysis rather than recomputed. The source_line table holds every line of every file, blank ones and comments included, so counting its rows would not give lines of code; the report is produced by the walk itself, which knows which lines carry code and which arrived from an include file.

Two details of that report matter. Its last row is a totals row named "Totals (across n files)" rather than after a file, so summing a column without excluding it doubles every figure. And a file which contributes a schema as well as code has more than one row in the file table, so the rows must be matched to the filtered set with an EXISTS rather than a join, or such a file's lines are counted once per row.

Using it

Generating the document

The generator is a command line tool, com.goldencode.p2j.report.server.GapDocumentDriver. Run it from the project's root directory:

```
java -DP2J_HOME=. -cp "p2j/build/lib/*" \
     com.goldencode.p2j.report.server.GapDocumentDriver [project] [output file]
```

Both arguments are optional:

Argument	Default	Purpose
project	The name of the project directory	Names the project in the document's title
output file	project_gap_analysis.textile, lower cased	Where the Textile is written, relative to the project root

So all three of these are valid, and the last two are equivalent:

```
java -DP2J_HOME=. -cp "p2j/build/lib/*" com.goldencode.p2j.report.server.GapDocumentDriver
java -DP2J_HOME=. -cp "p2j/build/lib/*" com.goldencode.p2j.report.server.GapDocumentDriver MyApp
java -DP2J_HOME=. -cp "p2j/build/lib/*" com.goldencode.p2j.report.server.GapDocumentDriver MyApp myapp_gap_ana
lysis.textile
```

System properties and classpath

P2J_HOME is the **project** root, not the FWD installation. It is what locates rptdb/, and it defaults to the current directory, so it can be omitted entirely when the tool is run from the project root. Set it when running from anywhere else:

```
java -DP2J_HOME=/path/to/project -cp "/path/to/project/p2j/build/lib/*" \
     com.goldencode.p2j.report.server.GapDocumentDriver MyApp
```

The classpath is FWD's own jars and their dependencies, which a converted project reaches through its p2j symlink at p2j/build/lib/. Nothing of the converted application is needed: the tool reads the report database, not the application.

No heap option is required for a project of ordinary size. A large project benefits from the same setting the report run uses, since the duplicate detection reads occurrence sets:

```
java -Xmx4G -DP2J_HOME=. -cp "p2j/build/lib/*" \
     com.goldencode.p2j.report.server.GapDocumentDriver MyApp
```

Exit status and output

The tool writes its progress to the console through the usual logging, ending with a line naming the file it wrote, how many items it found and how many need their marking reviewed:

```
INFO: Gap analysis: 96 candidate items
INFO: Gap analysis: 77 items after merging duplicates
INFO: Gap analysis written to /path/to/project/myapp_gap_analysis.textile: 77 items, 0 to review
```

It exits 0 on success and 1 if the report database could not be read or the document could not be written, with the cause logged.

Prerequisites

1. The analytics reports must already exist, which means a report run has been done for the project. If the gap marking rules have changed since that run, the conversion front end has to re-parse before the reports are regenerated: marking happens during the front end, which skips re-parsing when the 4GL sources have not changed, so a rules-only edit is otherwise ignored and the previous support levels are reported.
2. **The report web server must not be running.** H2 file-locks the report database, so the two cannot share it. This is not a limitation of the generator: the report run itself has the same constraint. Check with pgrep -f ReportWebServer and stop it first.

No server needs to be started. The driver opens the database itself.

What comes out

A Textile document with these sections:

- **Summary** — a count of items per group per support tier, which is the first thing to read.
- **Base Language**, **Database**, **User Interface** — the items, grouped by kind of feature, weakest support first, with conversion and runtime levels, use and file counts, and what is missing.
- **Windows-only facilities** — COM, ActiveX, window handles and .NET. The COM and ActiveX API surface is summarised by counts rather than listed, because a project which drives Excel reaches hundreds of members and no decision follows from reading the names one by one.
- **Statistics**.

From one test project:

Group	Unknown	No support	Partial support	Full, with restrictions	Total
Base Language	0	6	19	5	30

Database	0	6	11	6	23
User Interface	0	7	12	5	24
All groups	**0**	**19**	**42**	**16**	**77**

Reading an item

```
|_. Feature|_. Conversion|_. Runtime|_. Uses|_. Files|_. What is missing|
|can-find()|Partial|Full|34|17|Conversion: nested CAN-FIND inside a WHERE clause
converts to a client-side expression. Also reported by: Built-In Functions Called in
WHERE Clauses and Builtin Function Usage|
```

The two support levels are independent and both are shown: full conversion support is worth nothing if the runtime is stubbed, which is why the summary tiers are decided by the weaker of the two. "Uses" and "Files" say how much the project depends on the feature. "What is missing" is the gap detail recorded by the marking, and "Also reported by" names the reports which were merged into this item.

Adding marking for a new feature

Nothing new is required. Add the entry to the appropriate map in rules/gaps/ as before, and:

- if the map declares a group, there is nothing else to do;
- if the map is one of the five which mix groups, write the group on the entry with rw.gap();
- if there is something worth saying about what is missing, say it in the same call, as the conversion or runtime detail. That text is what fills the "What is missing" column.

Then regenerate the reports, forcing the front end to re-parse, and run the tool again.

When an item reports Unknown

That is a marking defect, not a finding about the feature. The feature reached a report which carries support levels, but no marking rule assigned it one. Find the map the feature belongs in and add it.

Current limitations

- The "What is missing" column is empty for most items. The marking rules carry roughly 658 end-of-line comments which explain the gaps, and they have not yet been migrated into rw.gap() details. CAN-FIND shows what the column looks like once they are.
- Built-in OO classes and methods can never carry gap details, because their levels come from a Java annotation which has nowhere to record them.
- There is no web application integration. The document is generated from the command line only.
- Round-tripping a reviewer's annotations back into a regenerated document is out of scope.

#6 - 09/13/2026 05:47 PM - Greg Shah

- Status changed from New to Test

- % Done changed from 0 to 100

#7 - 09/17/2026 08:56 PM - Greg Shah

Gap Analysis Document Generator: Performance and Presentation

Introduction

The gap analysis document generator (com.goldencode.p2j.report.server.GapDocumentDriver) was first delivered in revision 16746. Its first run against a large customer project took **twelve hours** and produced almost no output while it ran, so there was no way to tell whether it was working or hung.

This change makes the same run take **seventy six seconds** on the same data, and makes it report what it is doing while it does it. It also carries a set of presentation changes to the document itself: the statistics section moved to the front and gained four new measures, every heading is title cased, and every support level cell is colored the way the reports color it.

Checked in to branch 11747a, **revision 16768**.

The Problem

Two problems, one of them hiding the other.

The run took twelve hours. That is long enough that nobody would run the generator twice in a working day, which in turn means the document cannot be regenerated casually after a marking change — and regenerating it after a marking change is the whole point of having a generator rather than a hand written document.

The run was also silent. Three log lines were written in twelve hours: the candidate count at the end of the first phase, the deduplicated count at the end of the second, and the final summary. Between 17:50 and 03:03 nothing was written at all. There was no way to distinguish progress from a hang, and no way to tell which phase was expensive — which is exactly the information needed to fix the first problem.

Analysis

Method

Each phase of GapDocumentBuilder was extracted into a throwaway harness and run separately against the real report database, so that each query could be timed on its own rather than inferred from the gaps between three log lines. The harness established a report server session and applied the default file filter exactly as the driver does, so the queries ran against the same active_file contents.

The reference project measures 47,109 rows in the file table, 29,881 distinct source files, 11,910,796 lines of code and a 133 GB report database. The measurements below were taken with -Xmx16g.

Where the time went

Phase	Before	After
Read the gap categories	37 s	37 s
Merge duplicates	**2 h 45 m**	0.0 s
Break down how each attribute and method is used	**~9 h**	0.7 s
Read the features with no support level	3 s	3 s
Measure the project	~3 min	7 s
Total	**~12 h**	**76 s cold, 16 s warm**

Three of the five phases were never a problem. Reading the gap categories aggregates every match of every gap category in a single query and does it in under a minute; the presence reports take three seconds. The cost was concentrated in two phases, and a third contributed a few minutes.

The dominant cost: one query shape

The usage breakdown reads the parser dump of every occurrence of every attribute and method item, so that an item used on more than one kind of receiver can be broken down by receiver. It ran this query once per item:

```
select m.text, m.fid, m.line
  from match m
  join active_file a on a.fid = m.fid and a.sid = ?
 where m.cid = ?
```

```
 order by m.fid, m.line
```

Measured on the largest such category — 45,117 occurrences — that query takes **442.6 seconds**. There are 78 such categories on the reference project, which accounts for the nine hour block.

The reason is the join order H2 chooses. On the real table statistics it drives the query from active_file and reaches match through the foreign key index on match.fid, one file at a time, discarding everything whose cid is not the one asked for. The category's own occurrences are a rounding error next to the volume that plan reads: it walks essentially the whole match table for every category, and it does so in file order, which on a report database many times larger than memory is the worst possible page access pattern.

Rewriting the same question as an exists pins the driving table:

```
select substring(m.text, 1, 8192), m.fid, m.line
  from match m
 where m.cid = ?
   and exists (select 1 from active_file a where a.fid = m.fid and a.sid = ?)
 order by m.fid, m.line
```

The optimizer now enters through the index on match.cid, reads only that category's rows, and probes active_file once per row through the foreign key index it already has on fid. All 78 categories together: **2.6 seconds**, 95,197 rows, 13.5 million characters.

Why this did not show up earlier

Two things conspired to hide it.

It does not reproduce at small scale. On a synthetic database with 200,000 match rows against 47,000 active files, H2 picks match as the driving table for the join form as well, and both forms are equally fast. The bad plan only appears once the match table is large enough relative to active_file for the cost model to prefer the other order. Any benchmark built on a scaled down database would have shown nothing.

An EXPLAIN on a synthetic database misleads unless the schema matches exactly. active_file carries a foreign key to file, and H2 creates an index on the referencing column to enforce it. A synthetic reproduction built without that constraint produces a completely different plan, and led initially to the wrong conclusion that active_file had no usable index at all. It has one, on fid; what it does not have is one on sid, which is not what these queries need.

The other costs

Duplicate detection re-read every occurrence. Two reports describing the same gap are recognized by their occurrence sets being identical, since the reports name the same feature differently. Candidates were bucketed on a cheap fingerprint — occurrence count, file count and the minimum and maximum file id and line — and every bucket holding more than one candidate then had all of its members' occurrences read back and compared exactly. On the reference project that meant reading occurrence sets for categories with hundreds of thousands of matches each, through the same cold random reads: two hours and forty five minutes to confirm what the fingerprint had already all but established.

The lines of code totals asked the filter question once per cell. The totals are summed from the custom "Lines of Code Analysis By File" report, whose rows have to be restricted to the files the filter profile selects. That restriction was written as a correlated exists on each numeric cell, so the file name lookup ran once for every cell of every numeric column rather than once per row: **145.9 seconds**.

Report sizes counted rows the report run had already counted. reportSize() derived a report's occurrence count with count(m.id) over its matches. For the schema report "Fields by Table", that is 4,995,717 rows of cold reads — roughly six minutes — to arrive at a number the report run itself had already written into report_stats.

The Changes

Performance

1. **The usage breakdown drives from the category.** SQL_MATCH_TEXT replaces the join on active_file with an exists, as shown above. The occurrence's parser dump is also read through substring(m.text, 1, 8192) rather than in full: what the breakdown needs is the receiver, which is the dump's second line, plus the few lines a chained access walks through. The rest is the whole expression tree. On the reference project the longest of the 95,197 dumps read is 4,746 characters and the average is 142, so the cut never bites; an occurrence whose dump did not survive it would simply not be counted in the breakdown, which is already how the code treats a dump whose shape it does not recognize.
2. **Duplicate detection no longer reads occurrences at all.** SQL_ITEMS now computes two checksums over each category's occurrences in the pass it was already making: sum(mod(fid * 1000003 + line * 31 + col, 1000000007)) and sum(mod(line * 7919 + col * 131 + fid, 999999937)). Each term is reduced modulo a prime before it is summed, which keeps the total inside a bigint however many occurrences a category has. The two checksums join the counts and the bounds in the fingerprint, so candidates which share a fingerprint share an occurrence set and the exact comparison is no longer needed. The merge loop collapses accordingly: a bucket now yields exactly one survivor, the most specific report, with the others recorded against it.
3. **The lines of code totals filter by row.** SQL_LOC computes the set of surviving rowpos values once, in a subquery over the name column, and sums the numeric columns over that set with cell.rowpos in (...). The file name lookup stays an exists rather than a join, because a file which contributes a schema as well as code has more than one row in file and joining on the name would count its lines once per row.

4. **Report sizes read the report's own statistics.** SQL_REPORT_SIZE takes the occurrence count from report_stats.matches and counts only the categories, which is an indexed read. report_stats holds the unfiltered count, which is exactly what reportSize() has always returned, so this is a change of source and not of meaning.
5. **The source reports are measured in one query.** The include file report and the five user interface inventory reports are the largest reports the statistics touch. Asked one at a time they are six separate passes through a report database larger than memory; SQL_REPORT_USAGE asks for all of them at once, grouped by report title, and returns category, occurrence and file counts for each.

Progress reporting

Every phase now logs when it begins, and logs what it achieved and how long it took when it ends. Durations under a minute are written in seconds and longer ones as h:mm:ss. The usage breakdown, which is the only phase that reads a row per occurrence rather than a row per category, additionally reports every ten features. The statistics phase is broken into three reported sub-phases: the source and its lines, the includes and the user interface objects, and the schema. The driver logs what it is generating and where, names the filter profile it applied, and reports the total elapsed time alongside the item counts.

A representative run now looks like this:

```
INFO: Gap analysis of <project>, into /path/to/<project>_gap_analysis.textile
INFO: Gap analysis: file filter "Include All" applied
INFO: Gap analysis: reading the gap categories...
INFO: Gap analysis: 345 candidate items [36.8s]
INFO: Gap analysis: 272 items after merging duplicates [0.0s]
INFO: Gap analysis: breaking down how each attribute and method is used...
INFO: Gap analysis: usage breakdown, 70 of 78 features
INFO: Gap analysis: usage breakdown complete [0.7s]
INFO: Gap analysis: reading the features with no support level...
INFO: Gap analysis: features with no support level read [3.3s]
INFO: Gap analysis: measuring the project...
INFO: Gap analysis: counting the source and its lines...
INFO: Gap analysis: source counted [0.5s]
INFO: Gap analysis: counting the includes and the user interface objects...
INFO: Gap analysis: includes and user interface objects counted [29.8s]
INFO: Gap analysis: measuring the schema...
INFO: Gap analysis: schema measured [0.0s]
INFO: Gap analysis: project measured [30.3s]
INFO: Gap analysis written to ...: 272 items, 10 to review, in 76 seconds
```

Statistics

The section moved to the front of the document, immediately after the introductory note and before the summary. The figures are what the rest of the document is read against: a hundred partially supported features mean one thing in a twenty thousand line application and something else entirely in a twelve million line one, and a reader who meets the gaps first has nothing to weigh them against.

The temp-table schema no longer counts as a database. The schema reports carry it as a database of its own, which is what the reports of the web application want, since a temp-table is described by the same metadata as a permanent table and has to be filed somewhere. It is not a database the project connects to. SQL_DATABASES counts the categories of "Tables by Database" excluding the one named _temp.

Include file references say how far through the project they reach. The value is now written as 182,596 usages (in 24,645 files). How widely an include is used says more than how often: a hundred references from one program is a habit of that program, and the same hundred spread over a hundred programs is a dependency of the whole application.

Average program length is new: the total lines of code divided by the number of programs, where a program is one external procedure or one class, which is one source file either way. It is the crudest measure there is of how much work a feature woven through the application represents. Both figures it divides are rows immediately above it in the same table, so the reader can see what it is made of.

The user interface object inventory is new, drawn from the five reports added for it — Windows, Frames, Dialogs, Static Widget Usage and Dynamic Widget Usage — and presented in the same "usages in files" form, in the order a user meets them: the containers first, then what is placed inside them. A report database built before those reports exist simply has none of them, and the inventory is left out rather than reported as zero.

The section as it now renders:

Measure	Value
Source Files, .p	17,881
Source Files, .cls	10,529
Source Files, .w	1,471
Source Files, All Kinds	29,881
Lines of Code	11,910,796
Lines Written in the Program Itself	6,285,961 (52.8%)

Lines Arriving From Include Files	5,624,835 (47.2%)
Average Program Length	399 lines
Include Files Referenced	3,717
Include File References	182,596 usages (in 24,645 files)
Windows	3,375 usages (in 3,375 files)
Frames	9,040 usages (in 3,617 files)
Dialogs	92,459 usages (in 12,773 files)
Static Widgets	93,669 usages (in 4,893 files)
Dynamic Widgets	2,350 usages (in 688 files)
Databases	2
Permanent Tables	719
Temp-tables and Work-tables	6,664
Fields	4,995,717
Indexes	3,340
Sequences	102
Possible Gaps	272
Features Whose Marking Needs Review	10

Title cased headings

Every heading is title cased, as are the labels in the statistics table's measure column, so that the label column does not mix two conventions with the metric names in it.

The rule is conventional title case: every word is capitalized except the articles, the coordinating conjunctions and the short prepositions when they fall inside the title. Two details matter for this document in particular. A letter is only ever raised, never lowered, because the names here are 4GL and platform terms and COM/OCX, .NET, HWND and OO4GL are already written the way they are meant to be read. And only the first character of a word is considered, rather than the first letter found in it, so that a file extension stays .p rather than becoming .P.

Colored support level cells

Every support level cell in the item tables now carries the same background and foreground colors the reports use, so that a reader who knows the reports can see where a feature stands without reading the words. The colors are not duplicated: they are read from SupportLevelHelper.Descriptor, which is the single definition the reports and the wiki both describe, through the existing asRGBConstant() helper. A cell is written as Textile cell styling, for example:

```
|{background-color:#FF5000;color:#FFFFFF}. Stubs|
```

An unmarked feature keeps its own entry in that table, so an item whose level is LVL_UNKNOWN renders as a grey "Unknown" cell rather than as italic text.

Verification

The duplicate detection change was checked against the behaviour it replaces. Every fingerprint bucket holding more than one candidate had its members' exact occurrence sets read and compared, which is precisely what the old code did. **27 buckets, 27 confirmed identical, 0 false merges.** The buckets are the expected ones: a statement seen by "Language Statement Usage" and by the area report that describes it better, a built-in class seen both where it is named and where it is used, and a keyword reported both by its token and by its 4GL spelling.

The dump truncation was checked against the data. Of the 95,197 parser dumps the usage breakdown reads, exactly one exceeds 4,096 characters, at 4,746. The limit was set to 8,192 accordingly.

The statistics were checked for equality with the figures the slower code produced. Every figure carried over unchanged, including the 4,995,717 field count now read from report_stats rather than counted, which confirms that the two sources agree.

The end to end run was repeated after each change, against the same report database, and the document compared. Item counts, group counts and the review count are unchanged throughout.

Results

Measure	Before	After
Whole run, cold cache	~12 hours	76 seconds

Whole run, warm cache		16 seconds
Largest usage breakdown category	442.6 s	1.4 s
All usage breakdown categories	~9 hours	2.6 s
Duplicate detection	2 h 45 m	0.0 s
Lines of code totals	145.9 s	0.2 s
Log lines written during the run	3	21

Files Changed

File	Change
GapDocumentBuilder.java	The four query changes, the progress logging, and the new statistics
TextileGapRenderer.java	Statistics moved to the front, title casing, colored support level cells
GapDocumentDriver.java	Reports what is being generated, the filter profile applied, and the total elapsed time

No change was needed to DatabaseService, ReportApi or the report database schema. In particular, no index was added to active_file: the exists form rides the foreign key index the table already has on fid, and none of these queries filter on sid alone.

Notes for the Reader

The absolute timings in this note are specific to one report database on one machine, and the cold figures in particular are dominated by random reads against a store far larger than memory. What carries over to any project is the shape of the problem: the expensive queries were the ones asked once per category or once per cell, and each of the four changes replaces a repeated pass over the match table with a single pass or with a number already recorded.

The generator remains a command line tool. Nothing here changes how it is invoked.

#8 - 09/22/2026 04:25 AM - Greg Shah

Gap Analysis: Marking Corrections and Document Generation (r16781, r16782)

Introduction

Two revisions on branch 11747a, touching twelve files between them. They cover two strands of the same problem — **what the gap marking says about a feature**, and **how the generated document says it** — and they were committed together because several of the marking corrections were only visible once the document stopped obscuring them.

Revision	Committed	Files	Strand
16781	2026-09-20 12:59	8	Statistics split, metadata grouping, two mis-named features, one wrong default
16782	2026-09-20 15:47	8	Seven level corrections, the registry base key, and the document rework

The reference project throughout is dms: 29,881 source files, 11.9 million lines, a 133 GB report database.

Gap Marking Corrections

Levels which were wrong

Feature	Was	Now	Why
HWND	Full / Full (R)	**Full** / **Partial**	The attribute returns an FWD widget id rather than a window handle, which is deliberate. But it is meaningful only to the Win32 entry points FWD already emulates, and a project calling any other native API with it needs that emulation written first. That is work outstanding, not a permanent restriction
SYSTEM-HELP	Full / Full (R)	**Full** / **Partial**	FWD dispatches to an HTML resource at a configured service path and does not read or render .hlp or .chm files. Support for them could be added, so the gap is an implementation rather than something permanently out of reach
CPCASE, CPCOLL	Full / Partial	**Full** / **Stubs**	These move **down**. They were Partial on the strength of being readable from the startup parameters, but no configured value is honoured and none is reported back — the attribute reads as though nothing had been configured. Reading a value that nothing honours is not partial support
SECURITY-POLICY:GET-CLIENT	Full / Stubs	**Full** / **Full**	The implementation works. Its one deviation — it returns the live client-principal rather than a copy — is #11894, a defect in the implementation rather than a gap in the feature
OpenEdge.Net.URI:Decode(character)	Full / Stubs	**Full** / **Full**	The one-argument form passes no encoding, so the restriction which makes Decode(character, character) a stub — it ignores its encoding argument — does not reach it. The two-argument form is left alone
The four indexed Progress.Reflect.Variable accessors	Full / Partial	**Full** / **Stubs**	Get(index), Get(instance, index), Set(index, value) and Set(instance, index, value) are all UnimplementedFeature.missing bodies. They carried no marking of their own and so inherited the class's partial runtime, claiming support they do not have. See

			"The default which promoted four stubs" below

Progress.Lang.Enum reported as entirely unsupported

Every member of Progress.Lang.Enum read None / None, and the marking was not at fault in the way it looked.

The gap marking resolves a built-in OO class to the FWD Java class which implements it and reads the annotation there. Progress.Lang.Enum resolved to com.goldencode.p2j.oo.lang.Enum, which does not exist: the Java class is LegacyEnum, renamed because Enum collides with java.lang.Enum. Class.forName threw, and the marking recorded its class-not-found default of None / None. GetValue() had been marked full on LegacyEnum all along.

An entry in java_override_oo_names now maps it, in the same way Object, Class, Error and String were already mapped.

The default which promoted four stubs

A built-in OO member with no LegacyResourceSupport of its own now takes the level of the class which declares it. That correction is right for almost everything it touches, and wrong for four members.

Every LegacySignature member under com.goldencode.p2j.oo was read out of p2j.jar by reflection and every promoted member's Java body classified:

Level the member now reports	Body is an implementation	Body is a stub	Interface, no body
Full / Full	347	**0**	56
Full / Partial	50	**4**	5
Partial / Partial	22	0	0
Full (R) / Partial	5	0	0
Full / Basic	1	0	0

The 403 promoted to Full / Full are clean. The four are the indexed forms of Progress.Reflect.Variable, on a class marked CVT_LVL_FULL | RT_LVL_PARTIAL. They are now annotated as the stubs they are; re-audited, promoted members whose body is a stub went from **4 to 0**.

dms does not use any of the four, so this never reached its document. It would have reached every other project's.

Metadata tables were filed under the base language

The document's Base Language group carried a **Metadata Table** section listing eighteen tables — _file, _field, _index, _sequence, _dbstatus and the rest — while the metadata **field** rows immediately below them were correctly under Database. One kind of gap was being read in two places.

The marking map had declared the right group since r16746; the group was being lost on the way to the report. Half of that was already fixed. What remained was the fallback: a table the map does not name is marked None / None, and a fallback carried no group, so it fell to the documented Base Language default. Nine of the eighteen came through that hole.

A metadata table or field reference is now recorded as a database feature whether or not the marking map names it.

Reporting Corrections

ARGUMENTS is a preprocessor argument list, not a parameter list

The Statement Option section held one row:

Feature	Conversion	Runtime	Uses	Files
ARGUMENTS	None	None	381	361

Read plainly that says a project passing arguments to 381 RUN statements has no support for doing so. It is not what the row means. A RUN statement's parameter list is LPARENS, which is fully supported; ARGUMENTS is the **preprocessor** argument list, which only a system that compiles at run time can act on.

The report database holds the 4GL source line behind every match. All 381 read like this:

```
run add_error_message "You have to enter a sublet type.".
run add_error_message("Receive ID is empty.")).
RUN add_error_message("One or more error(s) ocurred while processing stock number " + QU...
run vp_message = "Prospect " + STRING(vp_prspid) + " does not exist".
```

Every one is a RUN whose parameter list is malformed — a missing pair of parentheses, an extra closing one, or an assignment where a call was meant. The 4GL parser takes the trailing tokens as preprocessor arguments, which is legal, so the code compiles and the parameters are silently not passed.

The row is now named **Preprocessor Arguments on a RUN Statement** and carries a conversion gap detail:

> If the OE project only runs in compiled form, then these are likely to be typos since they cannot work properly. In a development OE system or one that runs from source code, this could be valid code but cannot convert statically in FWD.

The level stays None / None, which is correct: FWD converts statically and cannot apply a preprocessor argument at run time.

The ActiveX controls were missing from the document

The section which holds everything whose presence alone is the gap said in its own introduction that it covered "COM automation, ActiveX controls and window handles" — and carried no list of ActiveX controls. The reports which find them had no featureType, which is the attribute that makes a report a source of gap analysis items.

OCX Control Loading by Descriptor now declares one, and the descriptor is reported without its 4GL string literal decoration — CtrlFrame rather than "CtrlFrame":U — the way the automation object report already reports a ProgID. On dms that is **622 controls** across 575 files.

One caveat, and it matters. This is a list of control instances, not of control types. The descriptor is the name a control is loaded under within its container, not the ActiveX class behind it. The control type is not recoverable from the analytics run at all: the ProgID and CLSID live in the .wrx file, which is binary OLE compound storage and is never parsed. Where the actual controls in dms have been identified — the ProEssentials graphing controls, the TList grid, the RMPHTML editor — that was done by reading the .ocx binaries by hand, outside the analytics.

Telling an INI file environment from the Windows registry

The 4GL environment statements — LOAD, USE, GET-KEY-VALUE, PUT-KEY-VALUE, UNLOAD — read and write either an INI file or the Windows registry, and **which one decides whether the use is portable at all**. An INI file is read the same way anywhere; the registry exists only on Windows. describe_registry reported only the keyword, so the Windows Registry Usage section said KW_LOAD, KW_GET_K_V and so on, and carried none of that.

LOAD is the only one of the five which names the store, through its BASE-KEY clause — confirmed from the grammar, where base_key_clause is used by load_stmt and nothing else. A LOAD is now reported under the base key it names:

- LOAD BASE-KEY INI (INI file)
- LOAD BASE-KEY HKEY_LOCAL_MACHINE (Windows registry)
- LOAD BASE-KEY (set at run time)
- LOAD (no BASE-KEY, platform default)

USE, GET-KEY-VALUE, PUT-KEY-VALUE and UNLOAD act on whichever environment a LOAD put in scope and cannot be attributed to one or the other from the statement alone. The section says so.

From a source count, dms should split roughly **18 registry against 7 INI**, plus one dynamic BASE-KEY p-baseKey.

Document Generation

The statistics became four tables

The document opened with a single thirty-row table carrying source counts, lines of code, include usage, the user interface inventory, the schema size and the gap count in one list — while the analysis below it is filed under three headings. A reader wanting to read the database gaps against the size of the schema had to find six schema rows among twenty-four others.

The statistics are now **Code Summary**, **Database** and **User Interface**, in the same order and under the same names as the groups the gaps are filed under, followed by **Gaps** — the former Summary table, which is a measure of the project like the others and is the one the rest of the document elaborates.

Possible Gaps and Features whose marking needs review were dropped from Code Summary, because the Gaps table carries both as its total and its Unknown column.

A row links to its own detail section

A feature with more than one kind of receiver gets a **How It Is Used** section further down. The table row now links to it, by a page-relative anchor so the link survives whatever the wiki page is eventually called:

```
|"@ERROR@":#usage-Base-Language-attribute-ERROR|...
h4(#usage-Base-Language-attribute-ERROR). ERROR &#8212; How It Is Used
```

The anchor is written explicitly rather than left to Redmine's heading-to-anchor rule, which the em dash in the heading would make unpredictable. It carries the group and the feature type as well as the name, because the same 4GL keyword can be an attribute of one thing and a method of another.

"Also reported by" is gone

A row with no gap details used to fill its "What is missing" column with the list of other reports the feature was seen in. That says nothing about the gap, and putting it where the gap details belong reads as though something had been recorded when nothing had.

Long tables collapse

A name table of more than twenty-five rows is wrapped in a {{collapse(Show Details)}} macro. On dms five sections collapse, the largest being the 622 OCX controls, which would otherwise turn one fact — how much of this is there — into twenty screens of scrolling. The rows are still there for a reader who wants them.

Features With No Support Level became Platform Specific Features

The old name stopped being true once the section took in features that convert and run perfectly well. Shared library calls and Windows registry usage are all marked Full / Full; what they cost is **portability**, not support. A native library has to exist on the new host, and a registry key does not exist off Windows.

Four sections were added, all read from reports which already existed:

Section	Source report	dms
Shell Command	Shell Commands	69 external programs the application launches
Shared Object/DLL API Call	Shared Object/DLL API Calls	355 entry points, ordered by name so the list reads as the set of native libraries — the categories are already LIBRARY:API()
Windows Registry Usage	Windows Registry Usage	5 rows, with the note explaining what the base key decides
Windows DDE Usage	Windows DDE Usage	**nothing.** dms has no DDE statements, so the section is dropped rather than printed empty

These are read **by report title**, not by featureType. That matters: featureType marks a report whose categories name a 4GL feature, and a shell command category is a command line the application happens to contain. The route by title keeps that distinction and, as a side effect, works against an existing report database without a re-run.

CONNECT statement parameters

A CONNECT statement converts and runs, so it carries no gap — but what a reader of the database group needs to know is which parameters the project actually passes, because that is what decides whether the connections can be expressed against a FWD database at all. An inventory table at the end of the Database group now lists them. dms passes -db and no-error and nothing else.

What This Changed in the dms Document

The document regenerates in **about fifteen seconds**.

	Before	After
Items	250	**228**
Base Language	110	86
Database	94	96
User Interface	46	46
Unknown / to review	0	0

The base language fall is the WHERE clause functions leaving and the metadata tables moving to Database; the database rise is those eighteen tables arriving, less what left.

A diff of the report database's feature categories before and after accounts for every change: **626 added** — 622 OCX controls, asc(), Hex Literal, the renamed ARGUMENTS row and one query option — **21 removed**, and **157 changed**, being 78 function rows, 51 metadata rows, 15 database triggers, 10 OO method and class rows and 3 others. Nothing changed that was not accounted for.

What Has Been Verified, and What Has Not

This is the part to read before relying on any level above.

The 2026-09-19 run was made against the r16781 content. So:

Change	State
Everything in r16781 — the metadata grouping, the ARGUMENTS naming, the OCX list, the Variable stubs	**Seen in a generated document**
Every generator change in both revisions	**Seen in a generated document** — regenerated from the same report database
Every marking correction in r16782 — HWND, SYSTEM-HELP, CPCASE, CPCOLL, GET-CLIENT, URI:Decode, Progress.Lang.Enum	**Not yet run.** The current document still shows the old levels for all of them
The registry base key reporting	**Not yet run.** The section still shows KW_LOAD

A conversion and report run is needed to confirm the second half. **The dms/p2j symlink points at a stale copy of the branch** and has to be repointed at ~/projects/11747a first, or the run will silently reproduce the old markings.

Short of a run, the following were verified directly:

- The Progress.Lang.Enum and Variable markings were read back out of the rebuilt jar by reflection.
- run_stmt_option_name was probed against a real dms AST and returns the new name on three ARGUMENTS nodes while leaving FILENAME, LPARENS and INT_PROC alone.
- The BASE-KEY navigation was probed against seven live nodes. **The first version of it was wrong** — the clause carries an EXPRESSION wrapper rather than the literal, so every LOAD would have been reported as "set at run time". That was caught by reading the AST rather than assuming its shape.
- ./gradlew jar and the testcases smoke test are clean on both revisions.

Open Questions

1. **Do Possible Gaps and Features whose marking needs review belong in the Code Summary table?** They measure the document rather than the project, and the Gaps table now carries both. They were removed; saying so here in case that is the wrong call.
2. **Does dms run only from compiled code?** If so, the 381 preprocessor argument uses are latent 4GL defects to be reported to the customer rather than a conversion gap.
3. **Is an instance-level OCX list what is wanted,** or should the work be done to name the actual controls by parsing the .wrx container or cross-referencing a hand-built map of the binaries?
4. **Is Full / Stubs right for the four indexed Progress.Reflect.Variable forms,** or is an indexed form better described as Full / None, being absent rather than incomplete?

What Is Not Established

- **The r16782 markings have not been seen in a generated document**, as set out above. The levels quoted for them are what the next run should produce.
- **Nothing here was run against OpenEdge.** Every statement about what FWD does is read from its source, its annotations or the report database.
- **The 381 preprocessor argument uses were not exhaustively classified.** Every one of the two dozen source lines read is a malformed call, but two dozen is not 381.
- **The OCX list answers "how much ActiveX is embedded and where", not "which controls"**, and cannot answer the second without work outside the analytics.
- **The audit behind the four Variable stubs covered com.goldencode.p2j.oo only**, because that is where the changed default reaches. Whether any other package has a class marked better than its members deserve was not asked.

Sources

- Branch 11747a r16781 and r16782
- #11894 — GET-CLIENT returns the live client-principal rather than a copy
- #11888, #11892 — the marking evaluation and the correction queue these came out of
- Generated DMS Gap Analysis — the generated document the counts are read from

#9 - 09/22/2026 09:32 AM - Alexandru Lungu

> Every item is filed under exactly one of three top level groups, in this reading order:

It would be ideal to have a standard hierarchy for any artifact around FWD: knowledge-base, testcases structure, reports, etc.

In Hierarchy there is also persistence, ui and base_language defined. This was inferred from test-cases project. But, there is also "runtime infrastructure" as a different category. This includes:

> AppServer connectivity (4GL-RPC, 4GL-WS, SOAP), the LOG-MANAGER and PROFILER session handles, the OE security model (AUDIT-CONTROL / AUDIT-POLICY / CLIENT-PRINCIPAL / SECURITY-POLICY / auth callbacks), and the ABLUnit framework itself.

Also, when working on that hierarchy, there was a decision to be taken for OO. In test-cases it is separate from these systems, but ultimately I decided to make it a "group" within the Base_Language system.