Progress®

Progress® OpenEdge® Memory Profiler In-Depth



Dustin Grau

Principal Software Engineer

November 2026

Session Objectives

- ✓ OEMP Background
- ✓ Challenges of Testing at Scale
- ✓ Using the OEMP Tooling APIs
- ✓ Establishing a Diagnostic Workflow
- ✓ Integration Opportunities (CI/CD and AI)



Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Who Am I?

MIA since 2020…did I miss anything?

- ✓ Assisted several customers with Progress® Application Server (PAS) for OpenEdge migrations since 2020
- ✓ Continued contributions on LTS features: 11.7 → 12.2 → 12.8
- ✓ Celebrated 24 years with Progress as of September ’26
- ✓ Daughter graduated from high school, now off to college!
- ✓ Still figuring out the empty-nester life…so I came (back) to PUG!

Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

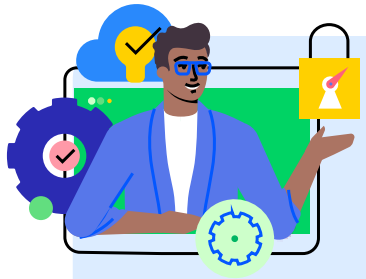
Where We've Been

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.



OEMP History

- Fall 2025: Introduced profiling capability in the AVM
- Winter 2025: Released the UI tooling using PAS for OpenEdge and a local DB with API layer
- Fall 2026: First tooling bugfixes and enhancements



Previous PUG Sessions

- **Jardosh**: Architecture and UI overview
- **Grau**: Pre-OEMP leak detection techniques
- **Chase**: UI-based analysis workflows
- **Wright**: Multi-tool workshop integration

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Gaps Addressed by OEMP

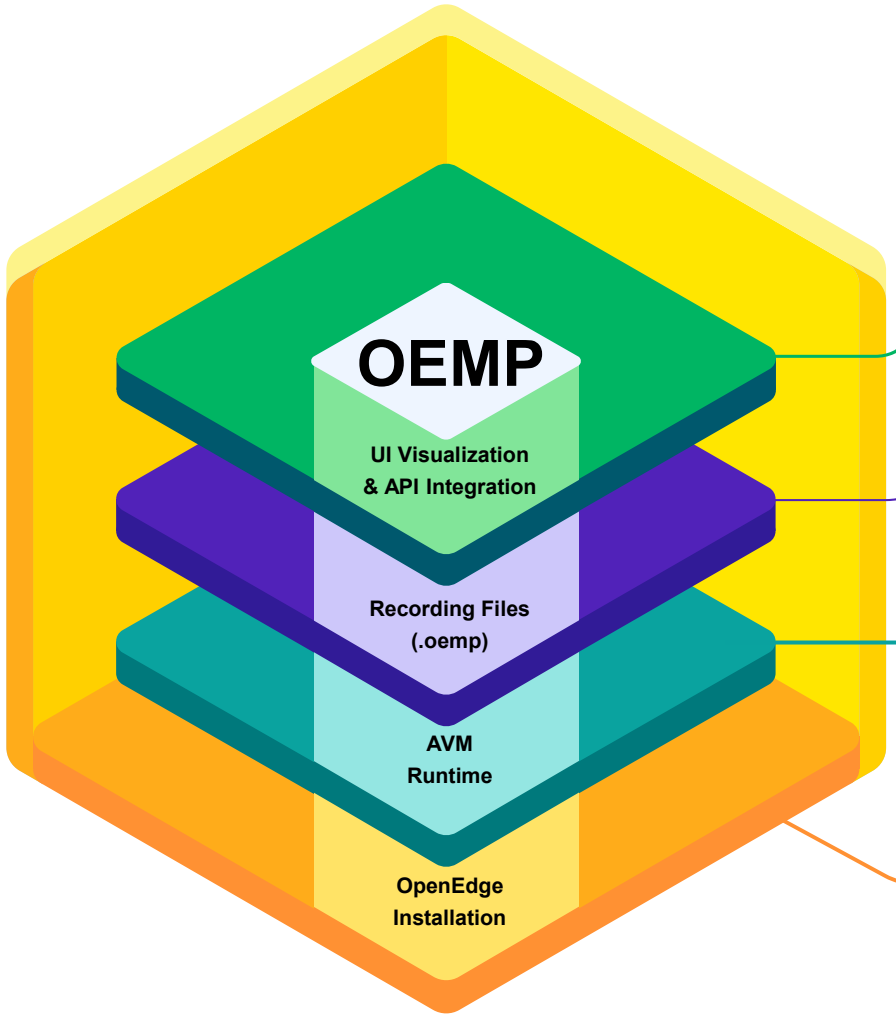
- ✗ **4GLTrace** as log-based memory leak detection was time-intensive and error-prone
- ✗ **ABLObjects Report** had no visibility into object lifecycle or detailed memory allocation
- ✗ Difficult and unfeasible to diagnose leaks in **production** PAS for OpenEdge environments
- ✗ Limited tooling and UI options for **proactive** testing and CI/CD integration

This Session's Unique Focus

- ✓ API-driven workflows at **scale** (managing high volumes of recordings)
- ✓ Integration opportunities: CI/CD pipelines and MCP-enabled AI **triage**
- ✓ Review file format internals for any **advanced** users (caveat emptor)
- ✓ Production triage strategies with **real-world** results for validation

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Memory Profiling Experience



UI Visualization and API Integration

- API Backend
- MCP Server Integration
- KendoReact Website
- CI/CD Process Integrations

Recording Files (.oemp)

- Direct File Import
- Streaming Data (Roadmap)

AVM Runtime

- PAS for OpenEdge
- ChUI/GUI

OpenEdge Installation

- OpenEdge 12.8.9+

Progress®
© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Where We’re Going

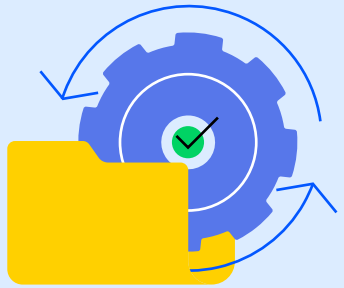
© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

1,000

.oemp files to review

1 PAS for OpenEdge Instance
1 ABL Application
20 agents × 50 sessions

"Manual review is impossible."



Real-world Test Scenario

- **Configuration:** 1 PAS for OpenEdge instance w/ 1 ABL application
  20 agents × 50 sessions = 1,000 ABL sessions (AVMs)
- **Load testing:** Consistent 5K virtual user load over 30-60 minutes
- **Hardware:** 64 vCPUs, 128 GB memory

An uncomfortable phrase: *"It only does this in production."*

- Can't reproduce in dev/test with 1-2 sessions
- Leaks are not always obvious in small-scale testing
- Production reveals cumulative effects across 1,000 sessions
- We built an internal application: APIs and structure for high-load testing

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Problem: Import Time Scaling

~16 hours

To import 1,000 recordings on a desktop-class machine

- Smallest File: 1.2MB, 46 snapshots, 5K Objects, with 30-minute test duration: ~60 seconds to fully import
- At Scale: 1,000 reports x 1 minute import = ~16 hours (possibly faster with better hardware, concurrency)

Human Factors

Make manual review at scale impractical

- Would you even recognize what you are looking for?
- Graph visualizations can only provide so much benefit before needing to exploring detailed data

- ✗ No objective criteria for “what's normal” vs “what's a leak” without established **baselines**
- ✗ Human fatigue leads to missed leaks or false positives
- ✗ By the time you finish reviewing, production has moved on

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Solution: Profiling as a Quality Gate

Strategies in this session…

- DART: Discover → Analyze → Refine → Trace
- Fast stress test + API-driven import/analysis
- Reviewing data from real-world validation
- Choosing effective configuration options

…lead to tasks you can perform!

- Use call stacks to guide you to a root cause
- Add leak-detection to nightly testing
- Use backend APIs for effective analysis
- Keep your import data to a minimum

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

When the Math Isn’t Mathing

Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Establish Your Baseline First

You can't detect abnormal without knowing what “normal” is

Your Baseline, aka. a “healthy” session

- Application Memory: Peak of ~17 MB (graph is essentially flat across timeline)
- Platform Memory: Peak of ~64 MB (graph is essentially flat across timeline)
- Session Memory: Peak of ~92 MB (via OEMANAGER Utility “status” report)

```
([AppMem] + [PlatMem]) + [N% Overhead] = Memory per Session
```

(17 MB + 64 MB = 81 MB) + 10.53 MB (avg. 13%) = **91.53 MB** (approx. session memory)

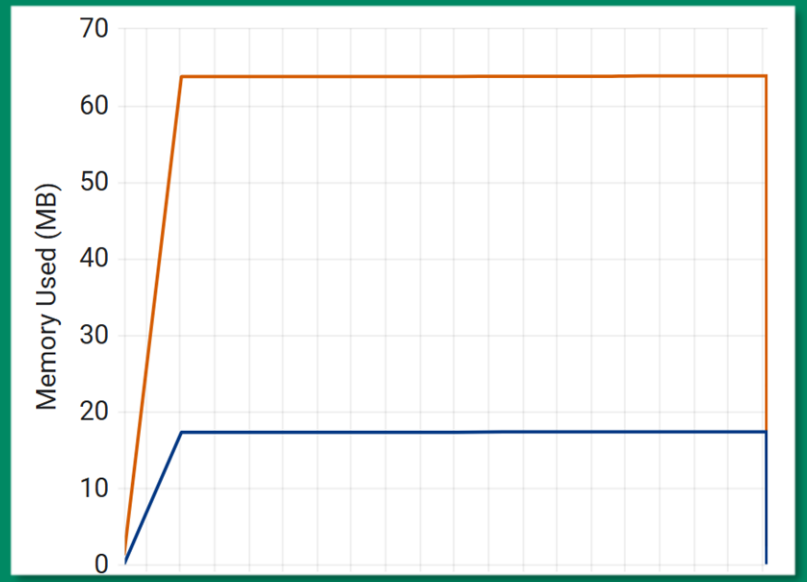
```
[Total # Sessions] × [Memory per Session] = Total ABLApp Memory
```

1 App@1,000 sessions×92 MB = **92 GB** (72% of 128 GB system just for PAS for OpenEdge)

YMMV: profile YOUR clean runs to establish YOUR baseline.



“Abby-someone… Abby-normal?”



Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved

When (and Why) Small Leaks Break the Math

30-Minute Test: A “Minor” leak

- Application Memory: 17 MB → 23 MB (+6 MB)
- Session Memory: 92 MB → 102 MB (+10 MB overall)
- Peak Objects: ~10K (**2x** baseline)
- Production Impact: 1,000 sessions × 10 MB = **+10 GB** unexpected (72%→80% of system memory)

✓ Baseline: 85 MB → 92 MB (+8%)
Leak: 92 MB → 102 MB (+**10%**)
Reason: Initial session startup + unusual excess

60-Minute Test: Scaling the Impact

- Application Memory: 17 MB → 41 MB (+24 MB)
- Session Memory: 92 MB → 126 MB (+34 MB overall)
- Peak Objects: ~28K (**5.6x** baseline)
- Production Impact: 1,000 sessions × 34 MB = **+34 GB** unexpected (72%→90%, OOM-killer looming)

✗ Baseline: 85 MB → 92 MB (+8%)
Leak: 92 MB → 126+ MB (+**36%+**)
Reason: Memory growth significantly scaled with time

Bottom line: Memory growth confirmed and the smaller test caught the bigger problem early

Progress®
© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Signal to Noise

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

How Many Snapshot Do I Need?

1. **Fewer snapshots still catch leaks: ~5-10 snapshots are enough for a minimal detection, under load**
   - Confirmed by testing: same leak detected with only 45 snapshots vs. 878 snapshots
   - Leaked objects will exist in `_PreCleanup` regardless of snapshot frequency

2. **For long-running profiles, defaults generate too much noise**
   - **ChUI/GUI**: consider **`cadence 300`** [seconds] (Default = 10)
   - *Example: 30-min test = 6 _AUTO snapshots vs. 180*
   - **PAS for OpenEdge:** consider **`cadence 0`** `w/` **`SnapshotEndOfRequest 100`** [requests] (Default = 10)
   - *Example: 10 snapshots per 1,000 requests to each ABL Session vs. 100*

3. **Bottom line, more snapshots ≠ better detection: more noise, not more clarity**
   - Default config (`SnapshotEndOfRequest=10`): larger files, slower import, messy analysis
   - Tuned config (`SnapshotEndOfRequest=100`): smaller files, faster import and analysis

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Configuration File Best Practices

- Use a named configuration file, e.g. “`mprofiler.config`”
  - Can add to a `startup.pf` file: `-profileMemory “/full/path/to/file/mprofiler.config“`
  - Store your config in source control with your code!
- Use a `description` field with meaningful values for reports
  - Bad description: "`test`" or left blank
  - Dynamically generate or replace as tokens in the file:
    - QA Testing suggestion: `description “<project>|<test_suite>|<PR-numbers>”`
    - Production suggestion: `description “<instance>|<hostname>:<port>|<version>”`
- Note: text will be truncated after 80-characters

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Anatomy of a Recording

Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Text-based, Incremental Snapshots

File Structure Essentials

Header: 1st _AUTO = recording metadata, UUID, version, timestamps, PID, session info, PROPATH

Snapshots: Contain any created, updated and deleted objects since the previous snapshot; has unique ID #

System: `_AUTO, _EndOfRequest, _PreCleanup, _Exit`
User: `Progress.Profiler.MemoryProfiler:TakeSnapshot("name")`

Call Stacks: Call paths for each object's creation

AVM Catalog: All known object types by OE version

Incremental Snapshot Model

- Snapshot = Diff since previous, not a complete picture
- Object lifecycles are reconstructed through the sequence of diffs, marking creation/deletion by snapshot ID
- Example:
  Snapshot 1 (+10),
  Snapshot 2 (+5, -3) = 12,
  Snapshot 3 (+8, -10) = 10 total objects
- Why this matters: Low snapshot frequency is fine as object deletions are still captured in the next diff

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Common Questions

What are critical snapshots for leak detection?

1. `_AUTO`: as the 1st snapshot, captured at profiler initialization, acts as an anchor point for growth
2. Ideally, an `_AUTO` or `_EndOfRequest` as the next snapshot, representing application initialization; consider this as your true starting point
3. `_PreCleanup`: before AVM termination, app leaks visible here (objects still present = potential leaks)
4. `_Exit`: after AVM cleanup, exposes true object lifetime (objects still present = AVM origin)

Couldn't I just parse the files myself?

- ✗ Report format could change without notice: if your parser breaks, it's your burden to fix
- ✓ Complex reconstructions offloaded: diff sequences, type grouping, lifecycle tracking, generation of call stacks
- ✓ We did the work: optimized queries with proper database indexes
- ✓ Tooling APIs remain in-sync with product code, just redeploy the OEMP instance from the DLC installation after upgrades

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Good Data, Bad Data

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Triage for Memory Leaks

DART: Discover → Analyze → Refine → Trace

D **Discover:** Files (`GetFileList`) vs. Recordings (`GetRecordings`)

- For a UI-based review of filenames with the same PID, look at snapshot count (higher) and file size (larger)
- For unimported reports (`GetFileList`): look at `StartTime, ElapsedTimeMS, FileSizeKB`
- For imported reports (`GetRecordings`): look for outliers in `HighWaterObjectCount` or `TotalAppObj`
- Key Point: You're scanning for **anomalies** among the recordings overall to move into the next steps

A **Analyze:** Memory Usage (`GetRecordingStatsSnapshots`)

- Returns time-series arrays for trend analysis of a given recording; the UI view plots a graph of data
- Compare memory values across the timeline: consistent growth = potential leak; flat/plateau = healthy

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

R **Refine:** Compare (`CompareSnapshots or GetObjectsTopN`)

- For observed growth: Use `CompareSnapshots` among snapshots before/after a rise in memory
  - Returns net changes by group (e.g. Classes, Procedures, etc.) with accumulations across objects
  - Positive values indicate overall growth by object type or name, depending on the group selected
- For exploring unknowns: Use `GetObjectsTopN` to narrow the objects by lifetime or type
  - Returns top objects either by longest lifetime (“LIFE”), memory size (“ALL”), or an object type ID
  - Repeated object types and patterns suggest systematic creations without proper cleanup
- Consider using both for a truly comprehensive view

T **Trace:** Objects in a Snapshot (`GetObjectStack`)

- Uses an Object ID from the **Refine** results to obtain the Call Stack to an object’s creation
- Navigate within your codebase add cleanup (e.g. FINALLY block), re-test and verify via profiling

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Common False Positives

- **The Bump + Plateau**
  - Evidence: Quick rise in memory, then stabilizes (IOW, not growing without bounds)
- **Expected Steady-State Objects**
  - Evidence: Registries, caches, static OOABL properties, connection pools (present w/o growth)
- **Objects in `_Exit` beyond `_PreCleanup`**
  - Evidence: Typically, database connections (persist until AVM termination)
- Platform Memory
  - Evidence: Essential components for ABL runtime (e.g. affected by `-s, -bt, -mmax, -D, etc.`)

Bottom line: If an object is ever-present but consistent in size/count, it's *likely* not of concern.

Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Shift Left: Test Early, Often

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The A/B Nightly Testing Workflow

1 Establish a Baseline

- Profile a known good release with a full regression suite; import to the OEMP tooling instance
- Document baseline metrics (app memory, peak objects, session memory) and note the recording ID

2 Nightly Test Execution

```
# Add profiling to existing nightly test
_progres -b -p nightlyRegressionTests.p -profileMemory "mprofiler.config"
```

- Tests run normally and new `.oemp` files are generated as build artifacts
- Sort and filter results, placing candidate files in the OEMP instance's import folder (manual or automated)

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

3 Automated Comparison (API-Driven)

- Import latest recording; fetch the baseline and latest recording information
- Compare `HighWaterObjectCount`: if latest > 2.0x baseline, trigger alert
- Key point: Compare metadata quickly, perform detailed analysis, only if flagged

4 Investigate Flagged Recordings

- If peak objects > 2x baseline: Halt deployment and perform detailed analysis
- Review memory trends via UI or API (`GetRecordingStatsSnapshots`)
- Review the call stack to understand the origin and potential root cause

5 Fix and Validate

- Apply fix to code; run same test with profiling
- Compare to baseline; verify peak objects back to range (<20% variance)
- If verified, approve for deployment; consider baseline update (if needed)

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Value Proposition

- ✓ **Time savings:** ~40 min test vs. weeks/months of production investigation
- ✓ **Cost savings:** Fix in dev ($) vs. production emergency ($$$)
- ✓ **Confidence:** Metrics are objective (nightly was 4x baseline) not subjective (“seems slower than usual”)
- ✓ **Prevention:** Find leaks before thousands of sessions multiplies the problem
- ✓ **Real-World Validation:** High-Load Testing w/Memory Profiling

Recording #1 (parameters: 5K users, 30 minutes)
5,086 peak objects, 17 MB app

Recording #2 (same params, 10% leak)
10,399 peak (2.0x over baseline) FLAGGED

Recording #3 (same params, 20% leak)
15,559 peak (3.1x over baseline) FLAGGED

Recording #4 (20% leak, now 60 minutes)
28,258 peak (5.6x over baseline) CRITICAL

Without nightly testing: 1 week = **1.9 TB** additional memory consumption, likely with OOM terminations
With nightly testing: issue flagged day 1, fixed before deployment

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Phone a Friend

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

AI-Assisted Triage via OpenEdge MCP Server

What is MCP?

- Model Context Protocol: Connects AI agents to data
- Translation: The AI agent gets access to YOUR data and knows how to interpret it based on the exposed “tools”

The Integration Opportunity

- ✓ Scenario: Of all recordings present, need to find leaks quickly
- ✓ Manual: Days of file-by-file review, subjective judgment, fatigue
- ✓ Natural language queries drive API-based analysis at scale
- ✓ Once set up, AI discovers and uses OEMP APIs automatically

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Art of Effective Prompts

Ineffective

- ✗ “Find memory leaks” = Non-specific
- ✗ “Check these recordings” = Not actionable
- ✗ “Is there a problem?” = Too vague

Effective

- ✓ “Analyze the available recordings to identify any potential trends which indicate a memory leak”
- ✓ “Compare the recordings from today's test run against a similar recording from last week's release”
- ✓ “Establish the most consistent recording as a baseline, and compare all other recordings to that one” ***
- ✓ “Which recordings show memory growth exceeding 20% above a baseline reading?”
- ✓ “For recording <X>, explain why the object count increased significantly”

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

OEMP + MCP + AI + U

What AI Handles Well

- Ranking and scoring across many recordings
- Pattern detection: Linear growth, stepped increases, plateaus
- Metadata comparisons: Peak objects, duration, memory deltas
- **Accelerating triage** and flagging of suspect areas of code

What Requires Human Judgment

- Validating whether detected patterns are actual leaks or considered as **expected behavior**
- Designing the fix and testing the solution **in context of your business needs**
- Making **release decisions** based on findings, scope and impact
- Understanding the **business logic and application architecture**

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Looking Ahead

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Changes for 12.8.12 and 13.1

1. OpenAPI Enhancements: Descriptions for all API operations, parameters and schema

- **Why?** Allows AI agents to more effectively discover the purpose of operations
- **Benefit:** Agents can chain operations intelligently through MCP integration
- **Bonus:** Not just for OEMP, this exists in the Netlib (OpenEdge.Net) library!

2. Support for importing partial or incomplete recording files

- **Why?** View recordings from AVMs terminated due to unexpected crashes or OOM-killer
- **Benefit:** Allows viewing of any in-flight recordings (those lacking an _Exit snapshot)
- **Caveat:** Without the _PreCleanup and _Exit the results will be inaccurate
  - IOW, not yet profiled far enough to capture all object deletions

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

The Practical Path Forward

- **Simplify configuration:** Basic config file + snapshot tuning = cleaner results
- **Baseline comparison pattern:** Establish your “normal” to find leaking code
- **Triage quickly with DART:** Discover → Analyze → Refine → Trace
- **Automation potential:** APIs enable CI/CD integration and AI triage
  - Reduce validation cycles to mere minutes from test start to actionable insights
- **Production math matters:** Small per-session leaks multiply catastrophically
  - Find leaks in testing, **not** production: testing-first is non-negotiable
- **You don't need to rewrite an entire application**
  - Start with high-traffic code that shows up often in the leak analysis
  - Re-test and compare results regularly to catch problems early
  - Confirm one code fix and move to the next!

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Questions?

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

News You Can Use

- Learn how to access the **OpenEdge AI Assistant**
- Get started with the **OpenEdge MCP Server**
- Book a demo of **Progress Agentic RAG**
- Modernize to **OpenEdge 12.8** today
- Explore what's new in **OpenEdge 13.0**
- Work with **Progress Professional Services** to accelerate adoption
- Learn about **security capabilities** across OpenEdge
- Explore the **developer tools** available for OpenEdge today
- Connect with us: **Discord, Community, CVP and more**



Progress®

© 2026 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.


Progress®