Notes from a 23-year codebase

· NetXMS · FrOSCon 2026

Five weeks ago

30 March 2026, 18:55

> *"Server crashes every second build on 32-bit ARM.*
> *Caught it under gdb but I don't yet understand."*

A message to my boss.

What gdb showed

```
2026.03.30 16:34:07.729 *D* [scheduler] Recurrent scheduler started
2026.03.30 16:34:07.730 *D* [scheduler] Registered scheduler task System.RegenerateAnomalyProfiles

Thread 137 "Scheduler/R" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x6cbb8de0 (LWP 22661)]
0x764e24b8 in wcslen () from /lib/arm-linux-gnueabihf/libc.so.6
(gdb) bt
#0  0x764e24b8 in wcslen () from /lib/arm-linux-gnueabihf/libc.so.6
#1  0x7649fa20 in ?? () from /lib/arm-linux-gnueabihf/libc.so.6
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

`wcslen()`. NULL. Corrupt stack. A 32-bit-only crash.

The bug

```
- nxlog_debug_tag(DEBUG_TAG, 7,
-   _T("InitializeTaskScheduler: added recurrent task %u at %s"),
-   task->getId(), task->getSchedule().cstr());
+ nxlog_debug_tag(DEBUG_TAG, 7,
+   _T("InitializeTaskScheduler: added recurrent task ") UINT64_FMT _T(" at %s"),
+   task->getId(), task->getSchedule().cstr());
```

`task->getId()` is `uint64_t`. The format string says `%u`.

On **64-bit**: doesn't crash by design — every variadic arg occupies one 8-byte slot, so `uint64_t` and `%u` consume the same slot. The value silently truncates to 32 bits, but task IDs fit, so output still looks right.

On **32-bit ARM**: `uint64_t` requires **two** slots (8 bytes, even-aligned), `%u` reads one. The next arg's offset shifts; `%s` reads NULL; `wcslen()` segfaults.

30 March 2026, 20:39

```
commit 465207173f
Author: Tatjana Dubrovica <zev@netxms.org>
Date:   Mon Mar 30 20:39:11 2026 +0300

    fixed wrong format specifier for uint64_t task ID
    in scheduler debug messages (crashes on 32-bit ARM)

 src/server/core/schedule.cpp | 30 +++++++++++++++---------------
 1 file changed, 15 insertions(+), 15 deletions(-)
```

1 hour 44 minutes from "I don't understand" to fix.

15 places in one file. The class of bug that only surfaces if you still build for 32-bit ARM in 2026.

Who's still on these platforms?

Two user bases, served by one engineering decision

Category	Platforms	Why they're still on it
Exotic	Solaris (SPARC + x86)	Specialist Unix in banks and industrial systems; never had a clean migration path
Old	RHEL 6, AIX 6.1, HP-UX, Windows 7	Migration costs millions; regulatory and audit constraints; service kiosks bound to specific hardware

Plus a smaller open-source segment — FreeBSD, OpenBSD, Alpine, 32-bit ARM — including the hardware that fired the crash from the opening.

They didn't choose to be a long tail

They chose **stability**.

Right to repair, for them, means:

> *What's working today should keep working tomorrow,*
> *even after the rest of the industry has moved on.*

The cost of the platforms we keep

One file, five backends

`include/nxatomic.h` — atomic primitives, custom-wrapped:

```
#ifdef _WIN32
   // native InterlockedExchange / _InterlockedCompareExchange64
#elif defined(__sun)
   // <sys/atomic.h>: atomic_swap_ptr, atomic_inc_32_nv
#elif defined(__GNUC__) && (__GNUC__ < 4)
   // inline assembly: lock; xaddl, xchgl
#elif HAVE_ATOMIC_BUILTINS
   __atomic_add_fetch(v, 1, __ATOMIC_SEQ_CST);
#else
   __sync_add_and_fetch(v, 1);  // older GCC fallback
#endif
```

One file. Zero external dependencies. Five compile-time paths.

A 5-year arc of one file

Year	What changed
2017-09	File extracted, `InterlockedExchangePointer` added
2017-12	HP-UX support added
2018	32-bit Windows fixes, Solaris build fixes
2020-01	**Solaris 9 removed**
2020-01	**PA-RISC removed**
2020-03	Preparation for AIX-on-xlclang
2020-07	Migrated `__sync` → `__atomic` builtins where available
2021-12	AIX: forced `__atomic` / `__sync` always
2022-08	**HP-UX removed entirely**

A living document of what we support *now*.

Who keeps these platforms alive

Three flavors of right-to-repair

Story	What kind of right-to-repair
Tribblix → Solaris SPARC OpenJDK	Community sustains a platform vendors abandoned
SWT GTK toolbar bug	A 14-year workaround, fixed by upstream collaboration
RAP menu order	Constraint can't be fixed upstream — we absorb it

Tribblix: April 2026 on Mastodon

We couldn't update Jenkins because Solaris SPARC didn't have a recent enough Java.

The answer came from **Peter Tribble** (Tribblix maintainer) and **Olaf Bohlen** — pointing to community-built JDK packages.

@zev333 @olbohlen Let us know how you get on! The sparc builds ought to be good enough for running Jenkins, although that's only a temporary workaround, as pushing the sparc openjdk port past the current jdk17 is going to be somewhere between difficult and impossible.

1

Olaf Bohlen favorited your post · Apr 2

Tatjana Dubrovica @zev333

@olbohlen @ptribble Thank you! I'll try!

Reply

Olaf Bohlen
@olbohlen@norden.social

Apr 2

@zev333 for Solaris @ptribble has builds on pkgs.tribblix.org/openjdk/spar... and pkgs.tribblix.org/openjdk/inte...

pkgs.tribblix.org

Solaris 11 SPARC jdk builds

15

"Difficult and impossible"

> *"The sparc builds ought to be good enough for running Jenkins,*
> *although that's only a temporary workaround,*
> *as pushing the sparc openjdk port past the current jdk17*
> *is going to be somewhere between difficult and impossible."*

— Peter Tribble, 2 April 2026

JDK 17 is the wall. After that, Solaris SPARC drops out of the modern Java ecosystem — unless someone keeps doing this work.

SWT issue #3236: A 14-year workaround

14 April 2026. GTK3 toolbar with `SWT.WRAP` causes a 60Hz repaint loop. 10–25% CPU while idle.

15 April: Lars Vogel digs out the original commit:

> *"The workaround was added on February 27, 2012 in commit 1c6100db3e by Arun Thondapu, as part of Bug 46025 — Toolbar does not support WRAP style. So the show_arrow toggle workaround has been there since the very first implementation of SWT.WRAP for GTK toolbars — over 14 years ago."*

16 April: *"Tested. Patch works. Thank you!"*

Two days. Three people. A wound nobody had revisited in 14 years.

RAP issue #346: When upstream can't fix it

November 2025. Menu event ordering bug in RAP.

Maintainer Ivan Furnadjiev:

> *"`e.doit = false` is not supported in RAP since many years as it requires synchronous requests, which are deprecated on some browsers and completely removed on others."*

The constraint isn't in RAP — it's in the **browser ecosystem above it**. RAP can't fix it. We carry a workaround in our codebase.

Right-to-repair sometimes means absorbing the immutability of layers above you.

Backward compatibility as discipline

Five layers of contract

The agent build from **February 2009** — still downloadable from our public release archive — still talks to the server we shipped last week.

How? Five distinct layers, each with its own discipline:

1. **NXCP wire protocol**
2. **Database schema**
3. **Configuration files**
4. **NXSL scripting**
5. **Export/import format**

Layer 1: NXCP wire protocol

The protocol between **agents and the server**. Versioned, with feature flags negotiated on connect.

- Commands are added, never removed
- New fields are optional in both directions:
  - **Agent → server:** absent from old agents; server handles the empty field
  - **Server → agent:** sent regardless; old agents ignore what they don't recognize
- Worst case: a feature added after the agent was built isn't available
- **The connection itself never breaks**

Layer 2: Database schema

The upgrade chain spans 20+ years. Every schema change ships with a migration script.

For very old installs, a stepping stone — written into the upgrade tool itself:

```
Your database has pre-2.1 schema (major version 0), which is no longer
supported by this tool. Install NetXMS 6.1 first, run `nxdbmgr upgrade`
there to bring the schema to a supported version, then upgrade to the
current NetXMS version.
```

src/server/tools/nxdbmgr/upgrade.cpp (April 2026, −11 990 lines)

Every reachable version has a documented path. Validated end-to-end in CI on every release.

Layer 3: Configuration files

A literal section in the agent config parser:

```
/* Parameters below are deprecated and left for
   compatibility with older versions */
{ _T("ActionShellExec"),              CT_STRING_LIST, ... },
{ _T("DataCollectionThreadPoolSize"), CT_LONG,        ... },
   // preferred: DataCollectionMaxThreadPoolSize
{ _T("EncryptedSharedSecret"),        CT_STRING,      ... },
{ _T("ExternalParameter"),            CT_STRING_CONCAT,... },
   // preferred: ExternalMetric
{ _T("ExternalParametersProvider"),   ...                 },
{ _T("ZoneId"),                       CT_LONG,        ... },
```

src/agent/core/nxagentd.cpp:420

Renaming is the default. Removal is the exception — usually security-forced (e.g., `EnabledCiphers` dropped in 6.1 with the AES-256 mandate).

Layer 4: NXSL scripting

Most NXSL changes follow a **standard deprecation cycle**: rename, document the new name, give a transition window with warnings — then the old name is removed.

For rare **structural** breaks, we ship a **conversion tool**.

Example: object access from `->` to `.` — after concatenation moved from `.` to `..` to free the operator up:

```
// Before:          // After:
node->name          node.name
```

When we have to break, we own the migration.

Layer 5: Export/import format

Last year: XML → JSON. Both parsers ship side by side.

```
// src/server/core/import.cpp:413
static ConfigurationFormat DetectConfigurationFormat(const char* content) {
   const char* ptr = content;
   while (isspace(*ptr)) ptr++;
   switch (*ptr) {
      case '{': return CONFIG_FORMAT_JSON;
      case '<': return CONFIG_FORMAT_XML;
      default:  return CONFIG_FORMAT_UNKNOWN;
   }
}
```

Three lines decide whether your 2024 export still imports.

Old XML → `import_legacy.cpp`. New JSON → `import.cpp`. **Both stay forever.**

We deprecate. We do it carefully.

Not "we never deprecate."

Not "we keep every feature forever."

But **every deprecation comes with a path forward** for the user who already has it working.

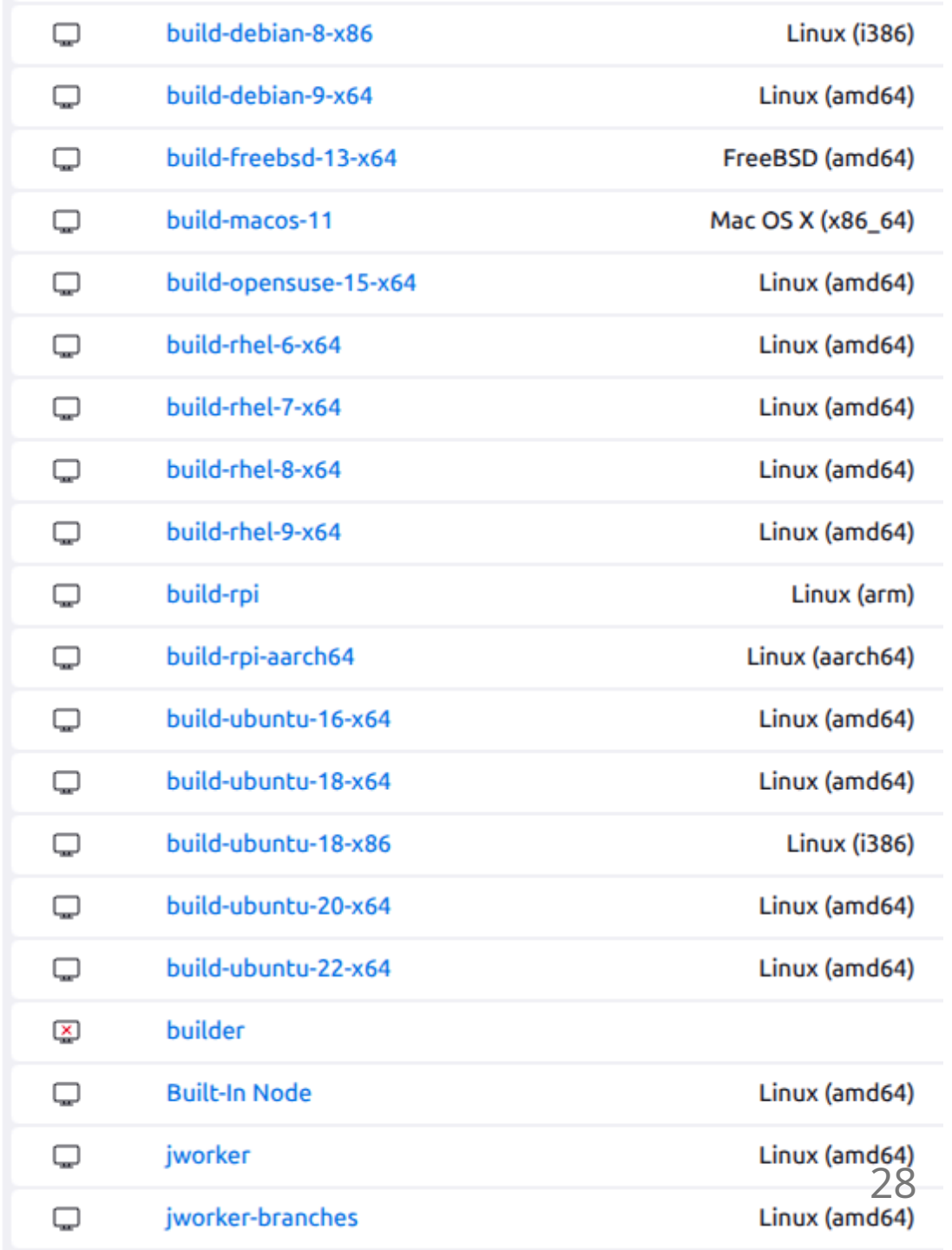
That's the contract.

What it costs · what it gains

Honest accounting

It costs:

- ~30 build agents to maintain *(see right)*
- Every PR risks *"AIX broke"* or *"SPARC broke"*
- CI catches **build** breaks. **Runtime** bugs (the ARM32 crash) slip through.
- C++11 ceiling — no `std::optional`, no `std::filesystem`
- Slower than Linux-only competitors



What it gains

- A user base that **chose stability** doesn't churn
- They pay for support, file useful bugs, show up on Mastodon to help
- Defensibility against Linux-only competitors who can't reach these customers
- A long-term commercial position that doesn't depend on hype cycles

A current parallel: Stop Killing Games

Europe's recent citizens' initiative passed first review:

> *Buyers shouldn't lose access to games they purchased*
> *when publishers shut down the servers.*

Open-source can't be killed by decree — even if a team folds, the code survives.

But it can be **left to rot**.

Software right-to-repair isn't only about source being available.
It's about keeping the contract intact long enough for users to actually rely on it.

citizens-initiative.europa.eu/initiatives/details/2024/000007

None of us is alone

What I'd ask you to take away

If you **maintain** something old:

> *You're not behind. You're doing work the rest of the ecosystem needs done.*

If you **depend on** something old:

> *You have leverage you may not realize. Tell maintainers what you need.*

Either way:

> *Keep talking — on Mastodon, in GitHub issues, in upstream PRs.*
> *Right-to-repair in software doesn't work alone.*

· NetXMS

@zev333@mastodon.social

linkedin.com/in/tatjana-dubrovica-zev

netxms.org