Qt and C++20

author: Marc Mutz

Principal Software Engineer, The Qt Company

speaker: Piotr Wierciński

Senior Software Engineer, The Qt Company

QtWS22
QtWS22
QtWS22
QtWS22
QtWS22

Oslo C++ Users Group

April 12, 2023

Disclaimer

This is not a C++20 presentation. Some familiarity with C++20 features is assumed in the audience. This is not the official roadmap. I’m not the Qt PM, just some guy that likes C++ and Qt. The overview is known to be incomplete. The views and opinions expressed in this presentation are my own and not those of my employer.Nothing you see or hear in this presentation is intended to be computer science advice. I’m not your software engineer and this is not software engineering advice. Please always do your own due diligence when it comes to programming and always take responsibility for your own choices.

Qt

Agenda

The Time Of Angels

What is C++20?

The Time Of Angels

What is C++20?

- 6th C++ Standard
- released Feb. 2020 (Prague)
  - __cplusplus == 202002L
- follows C++17
- precedes C++23

- new "major" release
  - major: C++98, C++11, C++ 20
  - minor: C++03, C++14, C++17, C++ 23
- major new features
  - modules, concepts, …

The Time Of Angels

C++20: a new language?

- C++11 brought:
  - Move semantics
  - Lambdas
  - constexpr
- “Feels like a new language”

- C++20 brings:
  - modules vs. header files
  - compile-time std::string and std::vector
  - concepts
  - ranges

Flesh And Stone

What's this gotta do with Qt?

Flesh And Stone

What’s this gotta do with Qt?

Qt is (still) implemented in C++

- C++20
  - deprecates
  - removes
  - adds

  features
- (some of) you *will* want C++20 mode for their projects

Qt needs to make sure

- Qt itself compiles
  - without warnings
- user code compiles
  - without warnings
    - caused by Qt headers
    - caused by Qt API use

Flesh And Stone

What's this gotta do with Qt? (cont'd)

Qt *will* require C++20:

- at some point
- we *must*
  - C++20 will be the default
- we *want*
  - new features for better APIs
  - reduce NIH

Nothing new:

- Qt < 5: any C++, no STL
  - MSVC 6, SunCC, anyone?
- Qt ≥ 5.0: C++98, *working* STL
- Qt ≥ 5.7: C++11
- Qt ≥ 6.0: C++17
- always with caveats

Flesh And Stone

Why should I care?

- QML-only? PySide-only?
- You don't!
  - might see execution speedups due to less data copying between JS/Python and C++ (NOI)

- *Some* C++?
- This talk is for *you*!

The Vampires Of Venice

What constrains Qt?

The Vampires Of Venice

What constrains Qt?

- binary compatibility
- source compatibility
- tool-chain availability

The Vampires Of Venice

Binary compatibility constraints

- app compiled against older Qt must run under newer Qt
  - cannot remove symbols
- within given patch release:
  - app compiled against newer must run under older(!) Qt
    - cannot *add* symbols

- without requiring C++20 we cannot use C++20 types in our ABI

The Vampires Of Venice

Source compatibility constraints

- app developed against older Qt must compile against newer Qt
  - cannot remove functions, types
- QUIP-6: SiC Category A
  - permissible SC breaks
- possible to deprecate stuff
  - tbr in next major release
- added QT_REMOVED_SINCE
  - allowing ABI and API to vary independently
  - except virtual functions
- added configure switch:
  - -disable-deprecated-up-to
  - cleans out obsolete ABIs
- rolling BC/SC window

The Vampires Of Venice

Tool-chain availability: C++17

- not all supported tool-chains fully conformant, yet
  - MSVC: lack of feature macros
  - libc++: lack of pmr, filesystem, BM
    - affects Apple, Android
  - QNX: GCC 8 (Qt requires GCC≥9)
  - MinGW: no std::thread/future
  - Integrity: std::wait_condition, …

- nothing new
  - Qt 5.15 still has MSVC std::atomic work-arounds
- expected: same with C++20
  - Qt will continue to use lowest common denominator of supported tool-chains
  - or use back-ports

The Vampires Of Venice

C++20 backport mechanism

- bad experience with backports of future std functionality so far
  - public API, so need to deprecate
  - Qt 5 qAsConst remained in 6.0
  - Qt 5 qExchange came with C++20 semantics, can't deprecate, yet
  - trend continues:
    - QExpected currently on Gerrit

- Solution: q*NN* mechanism:
  - re-implementation of std stuff
  - semi-private: use in pub headers
  - uses std impl where available
  - in namespace q*NN*
    - *NN*: C++*NN* semantics
    - q2[03]::exchange co-exist
  - user: pick lowest *NN* required
  - when Qt requires C++*NN*:
    - s/q*NN*::/std::/g, delete impl

Amy's Choice

What do C++20 features mean for Qt?

Amy's Choice

What do C++20 features mean for Qt?

- modules
- concepts
- coroutines
- comparison, 3-way
- constinit
- consteval
- constexpr additions

- <span>
- <version>
  - incl. feature-test macros
- <ranges>
- <chrono>
- <format>

(incomplete)

Amy's Choice

meta: Slide format

what feature is about

Qt context

Amy's Choice

Modules

- replacement for header files
  - faster compile times
- access protection mechanism
  - a first since C-with-classes
- promise to eliminate
  - -fvisibility=hidden
  - __declspec(export)

- macros can’t escape modules
- Qt uses a lot of macros
  - Q_OBJECT
  - Q_PROPERTY
  - Q_SIGNALS, Q_SLOTS
  - Q_DECLARE_METATYPE
- hopeless to modularize Qt any time soon

Amy's Choice

Concepts

- replacement for enable_if
- overload resolution
  - concept of refinement
    - (enable_if conditions must be mutually exclusive)
- terse template syntax
  - no `template <typename ~~~>`
  - easier for beginners
- verify that Qt types model relevant std concepts
- document Qt template constraints using concepts
  - avoid NIH while waiting for <concept>

Amy's Choice

Coroutines

- resumable functions
  - cooperative multithreading
  - lazy sequences
  - reactive programming
- three operations
  - co_yield (suspend+value return)
  - co_await (suspend)
  - co_return (exit CR)

- potential awaitables:
  - QFuture
  - timers
  - signals
  - QEvents
- NOIv2:
  - wean Qt off its containers
  - decouple API and impl

Amy's Choice

Coroutines (cont’d)

- C++20
  - just the language feature
  - only minimal lib support
- C++23
  - std::generator
- C++2c
  - executors, finally?

- how to avoid NIH?
  - QGenerator<T> might be acceptable
    - not owning vocabulary type
- QEventLoop as executor?
  - before C++2c: possible?

Amy's Choice

<=>

- "spaceship" operator
- 3-way comparison
  - think strcmp()
- returns one of three enums:
  - partial_ordering
    - allows unorderable values
  - weak_ordering
  - strong_ordering

- opp to unify impls & docs
  - all ops as hidden friends
  - documenting ordering type
- Qt vs. partial_ordering:
  - QOSVersion, QVariant(!)
  - expose unorderables?
    - probably
  - what about !isValid()?
    - discussing on dev ML atm

Amy's Choice

<=> (cont'd): generated operators

- C++20: stricter connection
  - between ==, !=
  - between <, >, <=, >=
- op==(A a, B b) generates
  - a != b, b == a, b != a
  - only need impl of op==(A a, B b)
  - errors if op==s are asymmetric
- op<=>(A a, B b) generates
  - a < b, a > b, a <= b, a >= b
  - b < a, b > a, b <= a, b >= a

- Qt would really benefit
  - but needs to support C++17
  - wait until we require C++20?
  - or hide behind macros?
    - discussing on dev ML atm

Amy's Choice

<=> (cont'd): structural types for NTTP

- op==/<=> can be = default'ed
  - member-wise equality
  - lexicographical ordering
- structural types
  - literal types with
  - = default'ed relational ops
  - only public members
  - can be used as NTTPs
- obvious candidates:
  - QFlags, qfloat16
  - QMetaType, QTypeRevision
  - QFixed (private API), . . .
- problem: only *public* members
  - fix: operator template (P2484)
    - still open, esp. not in C++23
    - *much* wider scope
  - may need to wait for C++2c :-(

Amy's Choice

constinit

- new C++20 keyword
- statically asserts
  - constant initialization
  - of static or thread_local objects
- important for Qt libraries
  - dynamically-initialized objects slow down start-up of every app using the library

- Q_CONSTINIT
  - already shipped in 6.4
  - available in C++17 (Clang only)
  - fixed a few dyn-inits in QtCore
  - roll-out ongoing
  - goal: all statics/thread_locals
    - constexpr, constinit, or
    - explicitly marked dynamic

Amy's Choice

consteval

- new C++20 keyword
- “immediate functions”
  - compile-time only
  - not callable at runtime

- applicability in Qt limited w/o “if consteval”
  - which is C++23-only

Amy's Choice

constexpr additions

- now constexpr in C++20:
  - std::vector
  - std::string
- dynamic memory allowed!
- "constexpr ALL the things"

- Qt types that would benefit:
  - QVersionNumber
  - ~~QString, QByteArray~~
    - not all-inline
    - atomic ref-counts
  - QVarLengthArray
    - conceivably

Amy's Choice

<span>

- generic view over
  - read-only (span<const T>)
  - or mutable (span<T>)
  - dynamically- (span<T>)
  - or statically-sized (span<T, 42>)
  - contiguous data

- decided to add QSpan
  - targeting 6.6 private, public later
    - when agreed how to use it
      - NOIv1
  - sans rvalue “bug” (impairs interop)
    - means QSpan may live forever
    - not an NIH issue, though:
  - *non-owning* vocabulary type, so no impedance mismatch with std::span

Amy's Choice

Feature Test Macros

- C++20 mandates SD-6 feature test macros now
- recognition that vendors ship new features piecemeal
- __cpp_xxx
  - language features, e.g.
  - __cpp_char8_t
  - predefined in each TU

- Qt has been requiring SD-6 feature test macros for post-C++11 features “forever”
  - MSVC didn’t support them
  - C++20 forced them to
  - benefits C++17 builds, too

Amy's Choice

Feature Test Macros: <version>

- __cpp_*lib*_xxx
  - library features, e.g.
  - __cpp_lib_char8_t
- SD-6: associated headers
  - <optional> for __cpp_lib_optional
  - __has_include needed
  - #include may #error in C++14
  - needed __cplusplus check
- C++20: all in <version>

- Qt doesn’t use <version>, yet
- availability:
  - Clang ≥ 7
  - GCC ≥ 9
  - no #error on use in C++17
  - MSVC: for two years now
  - QNX? Integrity? Apple?
- might be the first C++20 feature Qt will require

Amy's Choice

<ranges>

- replacement for STL algos
- takes a range instead of two iterators
- ranges are defined by
  - iterator/iterator
  - iterator/sentinel
  - incl. lazy sequences

- Qt containers already ranges
  - std-enable our views
    - problem: <ranges> include
    - (effect on compile times)
  - discussion whether COW containers should be views
    - std view concept allows this
    - but does it make sense?

Amy's Choice

<chrono>

- C++11 library
  - durations
  - clocks
  - time-points
- C++14 added UDLs:
  - 1s
  - 42ms
  - 1h + 30min // valid C++14!

- use more chrono in Qt APIs
  - some use in QtCore, but mostly
  - *int ms* (24d-ish range!)
  - or QDeadlineTimer
    - name: only for time_point?
    - requires one user-defined conversion more
    - granularity: s, ms or ns?

Amy's Choice

<chrono> (cont’d): C++20

- C++20 extends <chrono>:
  - day, month, year durations
  - timezone support
  - non-Gregorian calendars
  - leap years, seconds

- Qt has own API:
  - QTimeZone, QCalendar
- should interop with std API
  - partial support in 6.4
    - rudimentary
    - tacked on as afterthought
- might impl Qt with <chrono>
  - makes tzdb handling SEP

Amy's Choice

<format>

- C++20 adds Pythonic formatters:
  - format string like printf
  - extensible to user types like iostreams

- std::formattable Qt types
  - Qt types w/std equivalent
    - QString, QByteArray, …
    - same options as std types
  - types w/o std equivalent
    - which options to use?
- use <format> in
  - QDebug, QTestLib,
  - QX::toString(), QString::arg()

The Hungry Earth

Adoption of C++20:

The Hungry Earth

Adoption of C++20:

There will be two steps adoption of C++20 in Qt:

- Qt 6.9 will require C++20 for building Qt
- Qt 6.12 will required users of Qt to build with C++20

(given binary compatibility between stdlib versions)

The Hungry Earth

Key take-aways

Modularization far away even then

- Committed to C++20 compatibility (if feasible) for all active branches
  - warnings, hard errors, from headers and Qt API use
- Better, more consistent relational operators coming
- <ranges>, <format>, <chrono> support make sense (timeline tbd)
- Defined mechanism for std library backports (qNN)

The Hungry Earth

Outro

The Hungry Earth

Q&A

Questions?

Q&AQ&A
Q&AQ&A
Q&AQ&A
Q&AQ&A
Q&AQ&A

The Hungry Earth

References

- Qt's C++20 Jira epic: QTBUG-99243
- qNN explained: email on development ML
- Feature Test Macros: overview on cppreference.com
- NOIv1: Meeting C++ 2017 lightning talk
- NOIv2: Meeting C++ 2021 talk
- SiC Cat A: QUIP-6

The Hungry Earth

Thank you!