About me

• Senior Software Engineer, KDAB
• C++/Qt developer since ~2002
• Living in Limoges, France

Who doesn’t mind working on Windows?
Modernizing legacy MFC code
You got a big pile of MFC code...

... and now what?

- Continue as-is
- Abandon
- Rewrite
- Migrate

Focus
Reasons to move to Qt

• The MFC framework is on life support
  • Qt is supported, is improving, and has a lively community
• The UI feels very out-dated
• The application needs to run on multiple platform
• The application needs to follow new UI paradims (touch-screens...)
• It’s getting harder to add new features, fix bugs (technical debt)
• It’s getting harder to find good MFC developers
• ...

“It’s never the right time to move off MFC.”
Make it happen!
Rewrite vs Migration

+ Start from clean state
+ Not bound by technical decisions made decades ago
+ Revisit and clean up requirements
  - Long time to market
  - Lost years of knowledge, corner-cases
+ Keep the good, replace the bad
+ Migration can be done incrementally
+ Smaller time to market
  - Developers must **know both MFC and Qt**
  - Developers must **know migration techniques**

The Qt, OpenGL and C++ experts
Refactoring

• Avoid mixing migration and refactoring:
  • Increase development time
  • Prevent comparing old / new code
  • Make it harder to find bugs

• Advices for refactoring:
  • Refactor once the migration is done (or once the module is migrated)
  • Rewrite if the module is isolated from the rest of the code (partial rewrite)
What to migrate?

• **User interface**
  • Moving dialogs and UI elements to Qt equivalents.

• **Build system**
  • Moving from Microsoft-specific builds to a platform-independent build system

• **Non-UX code**
  • Moving the core MFC classes (like strings, files, containers...) to pure C++ or C++ with Qt as desired

• **Windows APIs**
  • Removing any Windows-specific APIs and replacing them with OS-generic equivalents
MFC Migration Techniques
Clean up your codebase

• Clean up warnings
• Document and simplify development setup
  • Make sure to have a reproducible setup
  • Make sure 3rd-party libraries are available
• Clean up warnings
• Remove dead-code
  • Use static analysis tools
• Clean up warnings
• (Optional) Beautify your code, update coding style
  • Automate the work with uncrustify, clang-format...
• Clean up warnings
Integrate Qt into the build system

- Download and install Qt Visual add-in
  - https://download.qt.io/archive/vsaddin/
- Add support of Qt add-in in your project
  - Edit project.vcxproj
  - Add QtVS_v301 keyword
  - Add qt_defaults property sheet
  - Add qt.props property sheet
  - Add qt.targets
  - Add QtSettings
- Decide how to handle PCH
  - moc supports adding an include
  - rcc does not!
Add QtWinMigrate to your project

- Qt/MFC migration framework
  - [https://github.com/qtproject/qt-solutions/tree/master/qtwinmigrate](https://github.com/qtproject/qt-solutions/tree/master/qtwinmigrate)
  - BSD license
- Contains 3 classes
  - **QMfcApp** – merge the Qt and MFC event loops
  - **QWinHost** – integrate native Win32 widgets/windows into Qt ones
  - **QWinWidget** – integrate Qt widgets into native Win32 widgets
Note on mixed MFC/Qt application

• QtWinMigrate allows to create mixed MFC/Qt application
  • The application is almost 100% feature complete during the port
  • Two issues with mixed MFC/Qt application:
    • Drag’n’drop from MFC to Qt (or vice-versa)
    • Focus handling

• 2 different approaches to migration

Avoid nesting too many levels of QWinHost / QWinWidget
Migrate dialogs

- UI descriptions are in the resource file (.rc)
  - Declarative description
  - “Easy” to parse
  - Different sections for:
    - Dialogs
    - Assets
    - Strings
    - Menus
    - Toolbars
    - …
Migrate dialogs – cont.

- Migrate the MFC message map into Qt paradigms
  - Qt event handlers
  - Qt slots (or normal functions)

BEGIN_MESSAGE_MAP(CTutorialDlg, CDialog)
  ON_WM_PAINT()
  ON_WM_TIMER()
  ON_WM_LBUTTONDOWN()
  ON_WM_MOUSEMOVE()
  ON_WM_RBUTTONDOWN()
  ON_WM_HSCROLL()
  ON_WM_VSCROLL()
  ON_BN_CLICKED(IDC_TIMER_CONTROL_SLIDERS,)
END_MESSAGE_MAP()

Qt event handlers
- void paintEvent(QPaintEvent *event);
- void timerEvent(QTimerEvent *event);
- void mousePressEvent(QMouseEvent *event);
- void mouseMoveEvent(QMouseEvent *event);

Qt slots
- void OnHScroll(int value);
- void OnVScroll(int value);
- void OnBtnClickedBtnAdd();
- void ...();

Automate message map migration
Migrate dialogs – cont.

- MFC dialog data exchange mechanism

**Dialog Object**
Class members

- `CString m_EchoText;
- CString m_HSliderEcho;
- CString m_VSliderEcho;
- CSliderCtrl m_VSliderBar;
- CSliderCtrl m_HSliderBar;
- CString m_MouseEcho;
- CString m_TimerEcho;

**Link objects in**
- `DoDataExchange`

**Initialize controls in**
- `OnInitDialog`

**Exchange data**
- `UpdateData`

- Qt does not need this complex mechanism
  - Use the widget pointers for the `ui` directly
  - Use Qt API for widget calls
  - Defer initialization using `showEvent`
Migrate other resources

• The resource files (.rc) contain:
  • Menus definitions
  • Toolbars definitions
  • Shortcuts (accelerators) definitions
  • Assets list
  • Strings translations
  • Languages, versions...

• During the migration, it’s important:
  • To automate the migration of resources
  • To be able to refactor the migrated code
Migrate controls

• Simple controls (checkbox, buttons...):
  • Use simple Qt widgets directly
  • Avoid any behavior changes (even if done in MFC)

• Complex controls (property grid, dock system...):
  • Usually from existing MFC libraries (ex: Toolkit Pro)
  • Create a new Qt widget with:
    • Same behaviour as MFC controls
    • Same (or close) API as MFC controls
  • Minimize changes in code during migration

• Use existing Qt libraries:
  • Advancing docking system: KDDockWidgets
We have barely scratched the surface

- Document / View architecture
  - CDocument, CView...
  - SDI, MDI...
- Printing and print previewing
- Threads
  - CWinThread...
- Component Object Model
  - COM, OLE, ActiveX
Conclusion
To sum up

- **Choose** your migration strategy wisely
  - Be prepared to defend it internally
  - Don’t mix migration and refactoring

- **Clean up** the code before anything

- **Use** the Qt/MFC migration framework
  - Don’t lose sight of the big picture: removal of MFC

- **Find** the right level of abstraction
  - Use existing libraries if they exist

- **Automate** all things:
  - resource migration
  - MFC code migration
  - project-specific code migration

- **Limit** code differences
Thanks!
nicolas.arnaud-cormos@kdab.com