Using C++11 to Improve Code Clarity: Braced Initialisers

David Rowland
Tracktion

Things a Programmer has to Consider

- Performance
  - Battery life
  - Responsiveness
  - Graphics for better UI/UX
  - Future improvements
- Readability
  - Coding styles
  - Clear intent
  - Other developers (and your future self)
- Maintainability
  - Time to change/refactor
- Reusability
  - Generic
  - Level of abstraction
- Robustness
  - Withstand future uses (threading)
- Security
  - Connections
  - Storing data
- Portability
  - Time to adapt to other platforms
  - Different UI form factors
- Compatibility
  - Fit with existing/future code
- Scaleability
  - From test cases to real world uses
  - Potential future uses

Solutions?

- Write less code

How?

- Type deduction (auto, decltype)
- Threads (std::async, std::future etc.)
- Lambdas
- Function objects (std::function)
- Variadic templates (parameter packs)
- Range based for loops
- Braced initialisers (std::initializer_list)

Braced Initialisers

- Object constructor deduction
- Aggregate initialisation
- Member declaration brace-or-equal initialisers

(C++17 P0134R0 proposes the term "default member initialiser")

Example 1: Returning Objects

```cpp
// C++98 (1)
std::pair<Path*, float> getPathForRow (int row)
{
    PredefinedGraphics& pg = *PredefinedGraphics::getInstance();

    switch (owner.getType (row))
    {
        case vstType:       return std::pair<Path*, float> (&pg.vstPath, 0.9f);
        case vst3Type:      return std::pair<Path*, float> (&pg.vst3Path, 0.9f);
        case auType:        return std::pair<Path*, float> (&pg.auPath, 0.9f);
        case rackType:      return std::pair<Path*, float> (&pg.rackPath, 0.7f);
        case internalType:  return std::pair<Path*, float> (&pg.internalPath, 0.5f);
    }

    return std::pair<Path*, float> (nullptr, 0.0f);
}
```

Example 1: Returning Objects (1)

```cpp
// C++98 (2)
std::pair<Path*, float> getPathForRow (int row)
{
    PredefinedGraphics& pg = *PredefinedGraphics::getInstance();
    Path* p = 0;
    float scale = 0.0f;

    switch (owner.getType (row))
    {
        case vstType:
            p = &pg.vstPath;
            scale = 0.9f;
            break;
        case vst3Type:
            p = &pg.vst3Path
            scale = 0.9f;
            break;
        case auType:
            p = &pg.auPath;
            scale = 0.9f;
            break;
        case rackType:
            p = pg.rackPath;
            scale = 0.7f;
            break;
        case internalType:
            p = &pg.internalPath;
            scale = 0.5f;
            break;
    }

    return std::pair<Path*, float> (p, scale);
}
```

Example 1: Returning Objects (2)

```cpp
// C++11 (3)
std::pair<Path*, float> getPathForRow (int row)
{
    PredefinedGraphics& pg = *PredefinedGraphics::getInstance();

    switch (owner.getType (row))
    {
        case vstType:       return {&pg.vstPath, 0.9f};
        case vst3Type:      return {&pg.vst3Path, 0.9f};
        case auType:        return {&pg.auPath, 0.9f};
        case rackType:      return {&pg.rackPath, 0.7f};
        case internalType:  return {&pg.internalPath, 0.5f};
    }

    return {nullptr, 0.0f};
}
```

Example 1: Returning Objects (3)

```cpp
// C++11 (4)
std::pair<Path*, float> getPathForRow (int row)
{
    PredefinedGraphics& pg = *PredefinedGraphics::getInstance();

    switch (owner.getType (row))
    {
        case vstType:       return {&pg.vstPath, 0.9f};
        case vst3Type:      return {&pg.vst3Path, 0.9f};
        case auType:        return {&pg.auPath, 0.9f};
        case rackType:      return {&pg.rackPath, 0.7f};
        case internalType:  return {&pg.internalPath, 0.5f};
    }

    return {};
}
```

Example 1: Returning Objects (4)

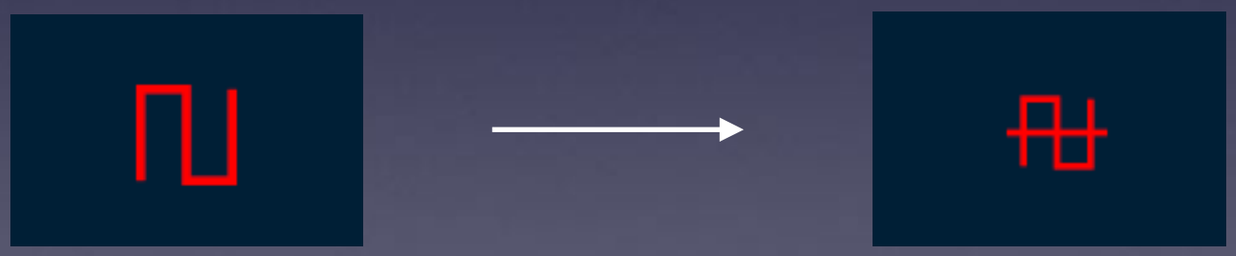
Example 2: Constructing Objects

```
void MainComponent::paint (Graphics& g)
{
    const Rectangle<float> bounds (getLocalBounds().toFloat());
    g.fillAll (Colour (0xff001F36));

    Path p (createSquareWavePath());

    const Point<float> p1 (-0.25f, 0.5f);
    const Point<float> p2 (1.25f, 0.5f);
    const Line<float> l1 (p1, p2);
    p.addLineSegment (l1, 0.1f);

    g.fillPath (p, p.getTransformToScaleToFit (
        bounds.withSizeKeepingCentre (20.0f, 20.0f), true));
}
```



Example 2: Constructing Objects (1)

```
// C++98 (1)
const Point<float> p1 (-0.25f, 0.5f);
const Point<float> p2 (1.25f, 0.5f);
const Line<float> l1 (p1, p2);
p.addLineSegment (l1, 0.1f);
```

```
// C++98 (2)
const Line<float> l1 (-0.25f, 0.5f, 1.25f, 0.5f);
p.addLineSegment (l1, 0.1f);
```

```
// C++98 (3)
p.addLineSegment (Line<float> (Point<float> (-0.25f, 0.5f), Point<float> (1.25f, 0.5f)), 0.1f);
```

```
// C++11
p.addLineSegment ({{-0.25f, 0.5f}, {1.25f, 0.5f}}, 0.1f);
```

Example 2: Constructing Objects (2)

```
PredefinedGraphics::PredefinedGraphics()
{
    Path glass (getMagnifyingGlassPath()), p;
    p.addPath (glass, glass.getTransformToScaleToFit ({1.0f, 1.0f}, true));

    p.addTriangle (1.2f, 0.3f, 1.6f, 0.3f, 1.4f, 0.6f);
}
```

```
// C++11
p.addTriangle ({1.2f, 0.3f}, {1.6f, 0.3f}, {1.4f, 0.6f});
```

Example 2: Constructing Objects (3)

Example 3: Using Initialiser Lists to Reduce the use of Temporary Arrays

```
ChannelBasedClass::ChannelBasedClass()
{
    Array<int> instruments;
    instruments.add (35);
    instruments.add (38);
    instruments.add (42);
    instruments.add (46);
    instruments.add (51);
    instruments.add (41);

    for (int i = 0; i < instruments.size(); ++i)
       addChannel (instruments.getUnchecked (i));
}
```

Example 3: Initialiser Lists (1)

```
// C++98 (1)
static int notes[] = { 35, 38, 42, 46, 51, 41 };

for (int i = 0; i < numElementsInArray (notes); ++i)
    addChannel (notes[i]);
```

```
// C++98 (2) - Adding a name
static int notes[] = { 35, 38, 42, 46, 51, 41 };

for (int i = 0; i < numElementsInArray (notes); ++i)
    addChannel (notes[i], MidiMessage::getRhythmInstrumentName (notes[i]));
```

```
// C++11 (3)
const auto notes = {35, 38, 42, 46, 51, 41};      // std::initializer_list<int>

for (auto n: notes)
    addChannel (n, MidiMessage::getRhythmInstrumentName (n));
```

```
// C++11 (4)
for (auto n: {35, 38, 42, 46, 51, 41})
    addChannel (n, MidiMessage::getRhythmInstrumentName (n));
```

Example 3: Initialiser Lists (2)

Example 4: Member Declaration Brace-or-Equal Initialisers

```
// C++98 (1)
class MidiNote
{
public:
    MidiNote()
        : startBeat (0.0), lengthInBeats (0.0),
          noteNum (0), chan (0), velocity (0),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (double startBeat_, double lengthInBeats_,
              int noteNum_, int chan_, int velocity_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat, lengthInBeats;
    int noteNum, chan, velocity;
    int colourIndex;
    int noteID;
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (1)

```
// C++98 (2) - Adding a ValueTree constructor
class MidiNote
{
public:
    MidiNote()
        : startBeat (0.0), lengthInBeats (0.0),
          noteNum (0), chan (0), velocity (0),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (double startBeat_, double lengthInBeats_,
              int noteNum_, int chan_, int velocity_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v]),
          colourIndex (0)
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat, lengthInBeats;
    int noteNum, chan, velocity;
    int colourIndex;
    int noteID;
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (2)

```
// C++98 (2) - Adding a ValueTree constructor
class MidiNote
{
public:
    MidiNote()
        : startBeat (0.0), lengthInBeats (0.0),
          noteNum (0), chan (0), velocity (0),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (double startBeat_, double lengthInBeats_,
              int noteNum_, int chan_, int velocity_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v]),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat, lengthInBeats;
    int noteNum, chan, velocity;
    int colourIndex;
    int noteID;
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (3)

```
// C++98 (3)
class MidiNote
{
public:
    MidiNote()
        : startBeat (0.0), lengthInBeats (0.0),
          noteNum (0), chan (0), velocity (0),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (double startBeat_, double lengthInBeats_,
              int noteNum_, int chan_, int velocity_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v]),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat, lengthInBeats;
    int noteNum, chan, velocity;
    int colourIndex;
    int noteID;
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (4)

```
// C++11 (4) - Using brace-or-equal initialisers
class MidiNote
{
public:
    MidiNote() {}

    MidiNote (int noteNum_, int chan_, int velocity_,
              double startBeat_, double lengthInBeats_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_)
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v])
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat {0.0}, lengthInBeats {0.0};
    int noteNum {0}, chan {0}, velocity {0};
    int colourIndex {0};
    int noteID {getNextNoteID()};
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (5)

```
// C++98
class MidiNote
{
public:
    MidiNote()
        : startBeat (0.0), lengthInBeats (0.0),
          noteNum (0), chan (0), velocity (0),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (double startBeat_, double lengthInBeats_,
              int noteNum_, int chan_, int velocity_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v]),
          colourIndex (0),
          noteID (getNextNoteID())
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat, lengthInBeats;
    int noteNum, chan, velocity;
    int colourIndex;
    int noteID;
};
```

```
// C++11
class MidiNote
{
public:
    MidiNote() {}

    MidiNote (int noteNum_, int chan_, int velocity_,
              double startBeat_, double lengthInBeats_)
        : startBeat (startBeat_), lengthInBeats (lengthInBeats_),
          noteNum (noteNum_), chan (chan_), velocity (velocity_)
    {
    }

    MidiNote (const ValueTree& v)
        : startBeat (v[IDs::s]), lengthInBeats (v[IDs::l]),
          noteNum (v[IDs::n]), chan (v[IDs::c]), velocity (v[IDs::v])
    {
    }

    // accessors/mutators
    // ...

private:
    double startBeat {0.0}, lengthInBeats {0.0};
    int noteNum {0}, chan {0}, velocity {0};
    int colourIndex {0};
    int noteID {getNextNoteID()};
};
```

Example 4: Member Declaration Brace-or-Equal Initialisers (6)

Summary

- Use C++11 braced initialisers to reduce the amount of code
  - Object constructor deduction
  - Aggregate initialisation
  - Member declaration brace-or-equal initialisers

- Less code means:
  - Quicker to write
  - Quicker to read
  - There’s less to reason about
  - Clearer intent
  - More robust, maintainable
  - Likely to be more optimisable