C++ now

2025

Techniques for Declarative Programming in C++

Richard Powell

Richard Powell

rmpowell77@me.com

BEATNIK

Richard Powell

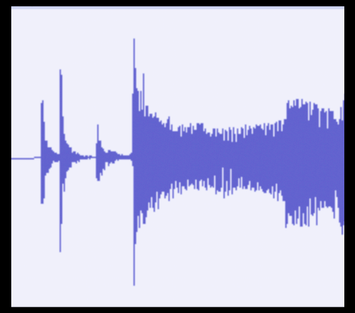
rmpowell77@me.com

Richard Powell

rmpowell77@me.com

Richard Powell

rmpowell77@me.com



Richard Powell

rmpowell77@me.com

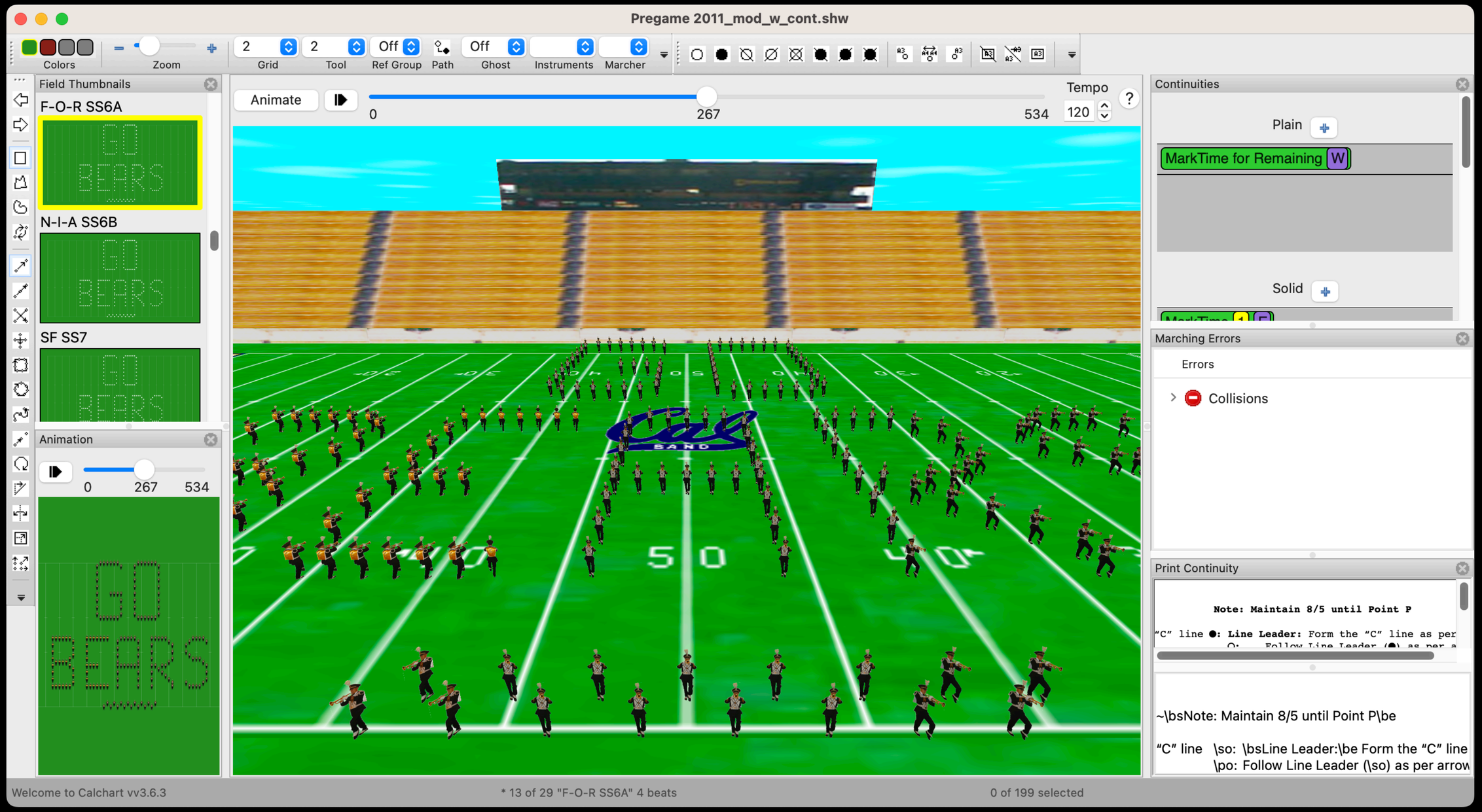
- A case study in Declarative Programming

- A case study in Declarative Programming
- Practical techniques you can use

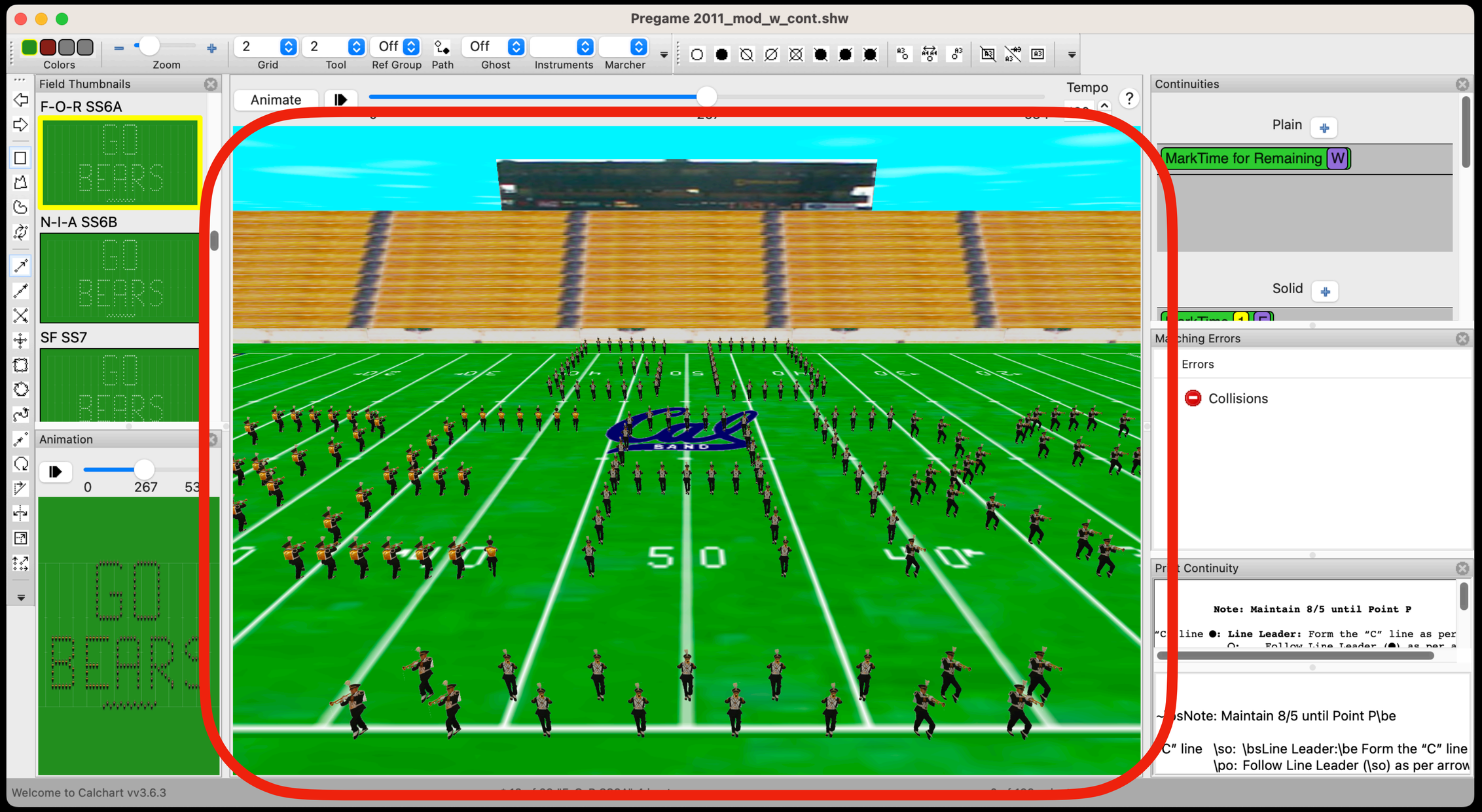
- A case study in Declarative Programming
- Practical techniques you can use
- We're going to be working with GUIs, but this is not a talk on UIs.

My journey

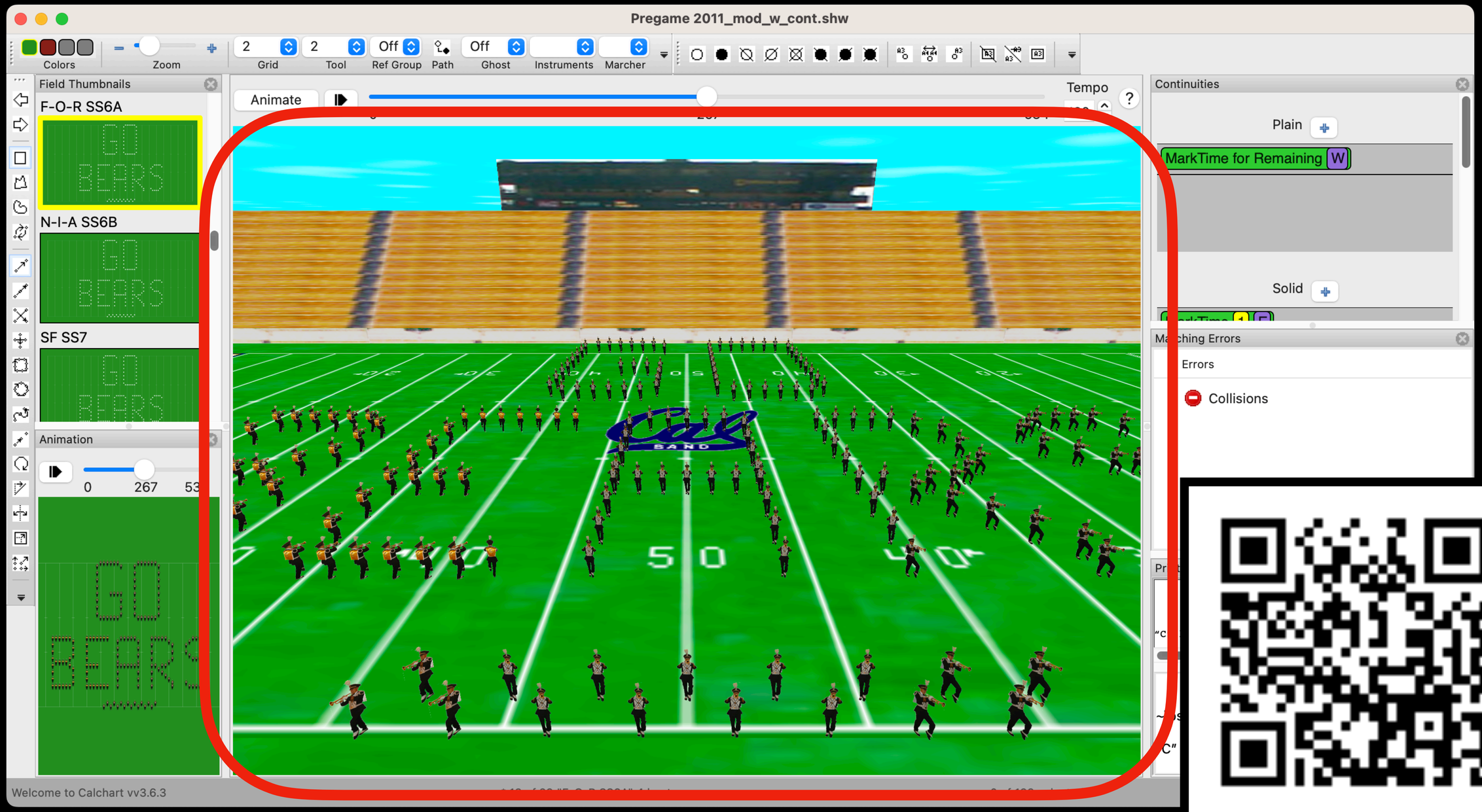
My journey



https://github.com/calband/calchart



https://github.com/calband/calchart



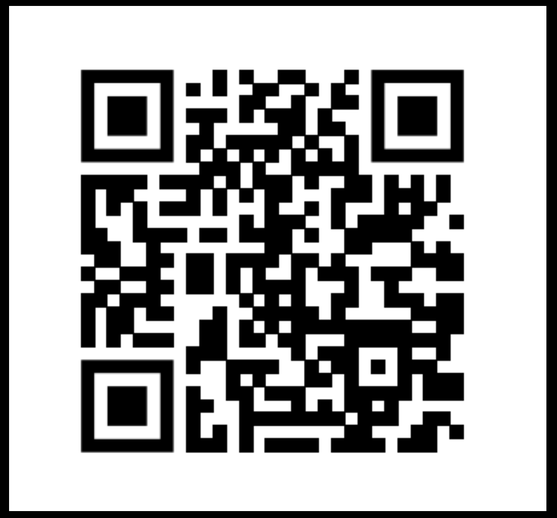
https://github.com/calband/calchart

wxWidgets is a C++ library that lets developers create applications for Windows, macOS, Linux and other platforms with a single code base. It has popular language bindings for Python, Perl, Ruby and many other languages, and unlike other cross-platform toolkits, wxWidgets gives applications a truly native look and feel because it uses the platform's native API rather than emulating the GUI. It's also extensive, free, open-source and mature.

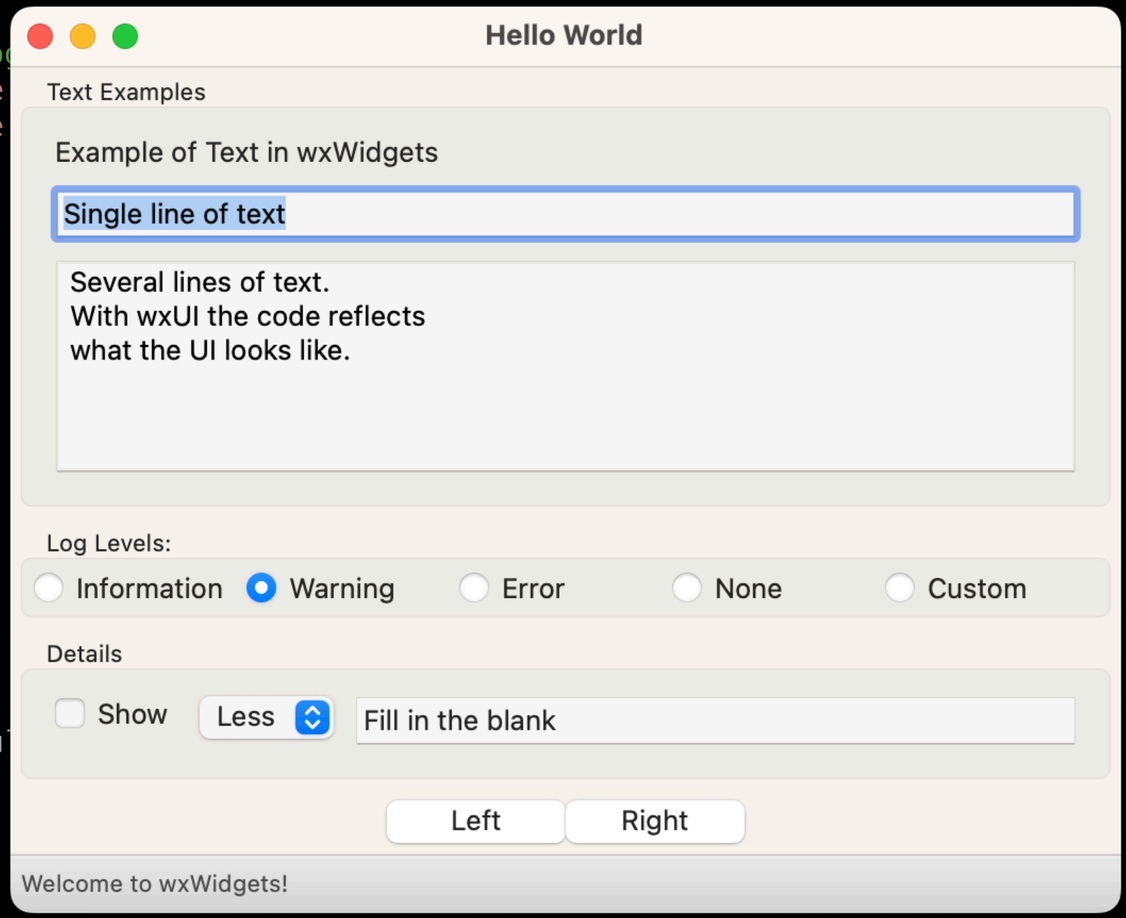
Let’s make an App!

Demo time

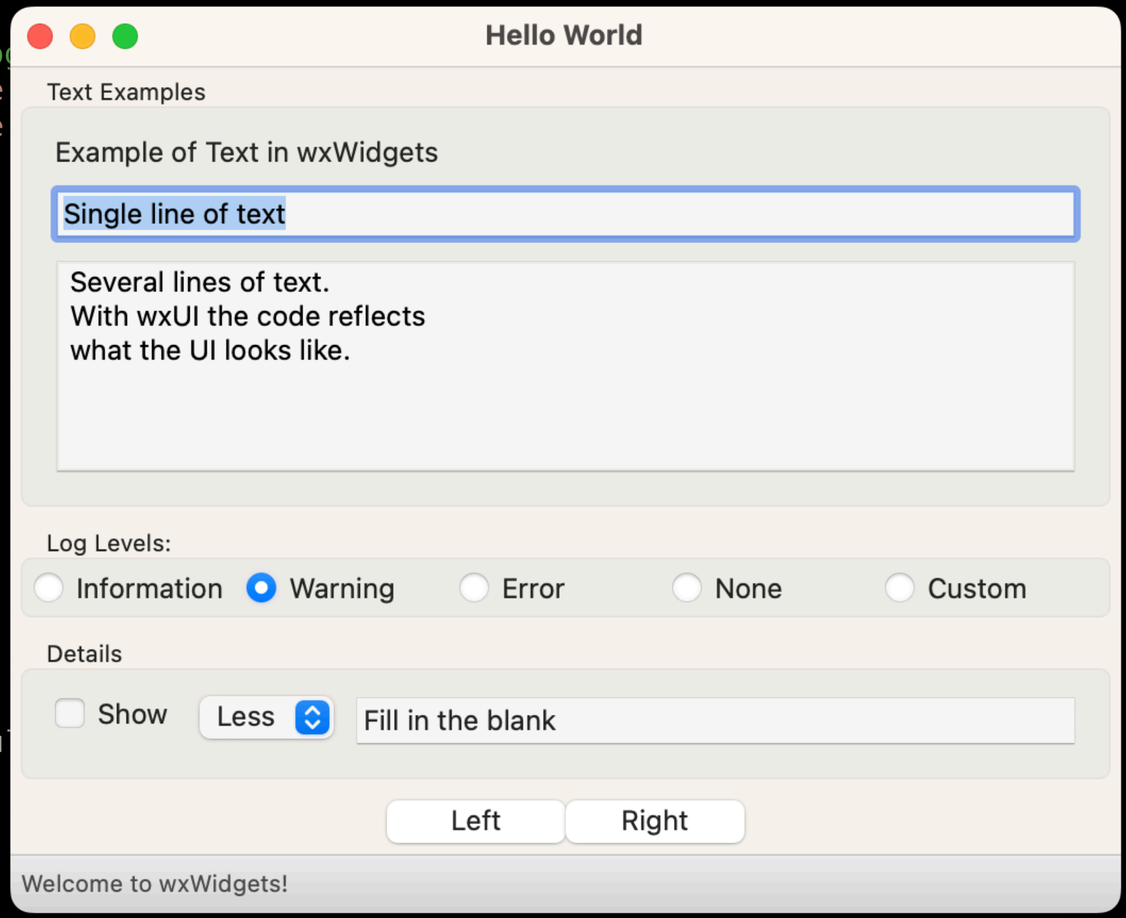
https://github.com/rmpowell77/wxHelloWorld



```
SetMenuBar(menuBar);

CreateStatusBar();
SetStatusText("Welcome to wxWidgets!");

// Create the controls.  This is cribbed from the RichTipDialog
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Example
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Single
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefau
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls
```

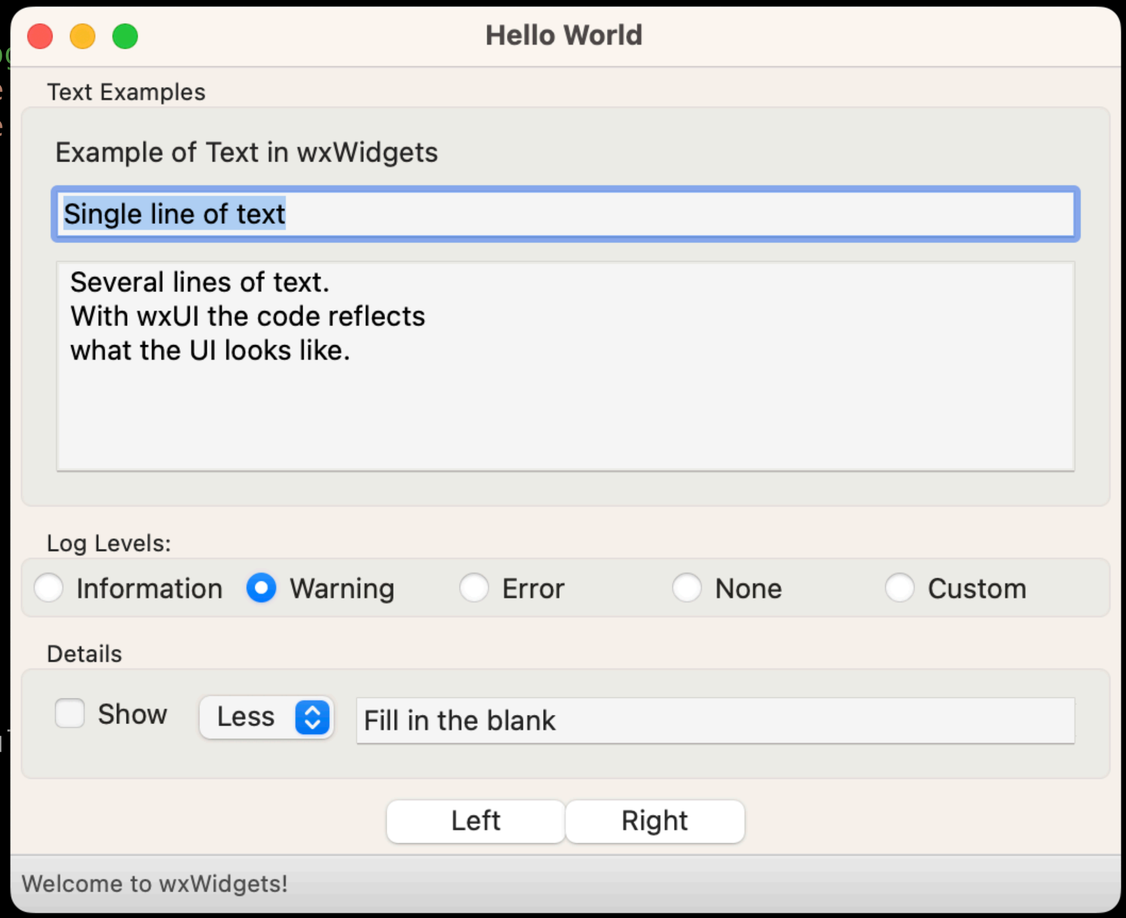


```
SetMenuBar(menuBar);

CreateStatusBar();
SetStatusText("Welcome to wxWidgets!");

// Create the controls.  This is cribbed from the RichTipDialog
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Example
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Single
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefau
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls
```



```
SetMenuBar(menuBar);

CreateStatusBar();
SetStatusText("Welcome to wxWidgets!");

// Create the controls.  This is cribbed from the RichTipDialog
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Example
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Single
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefau
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls
```

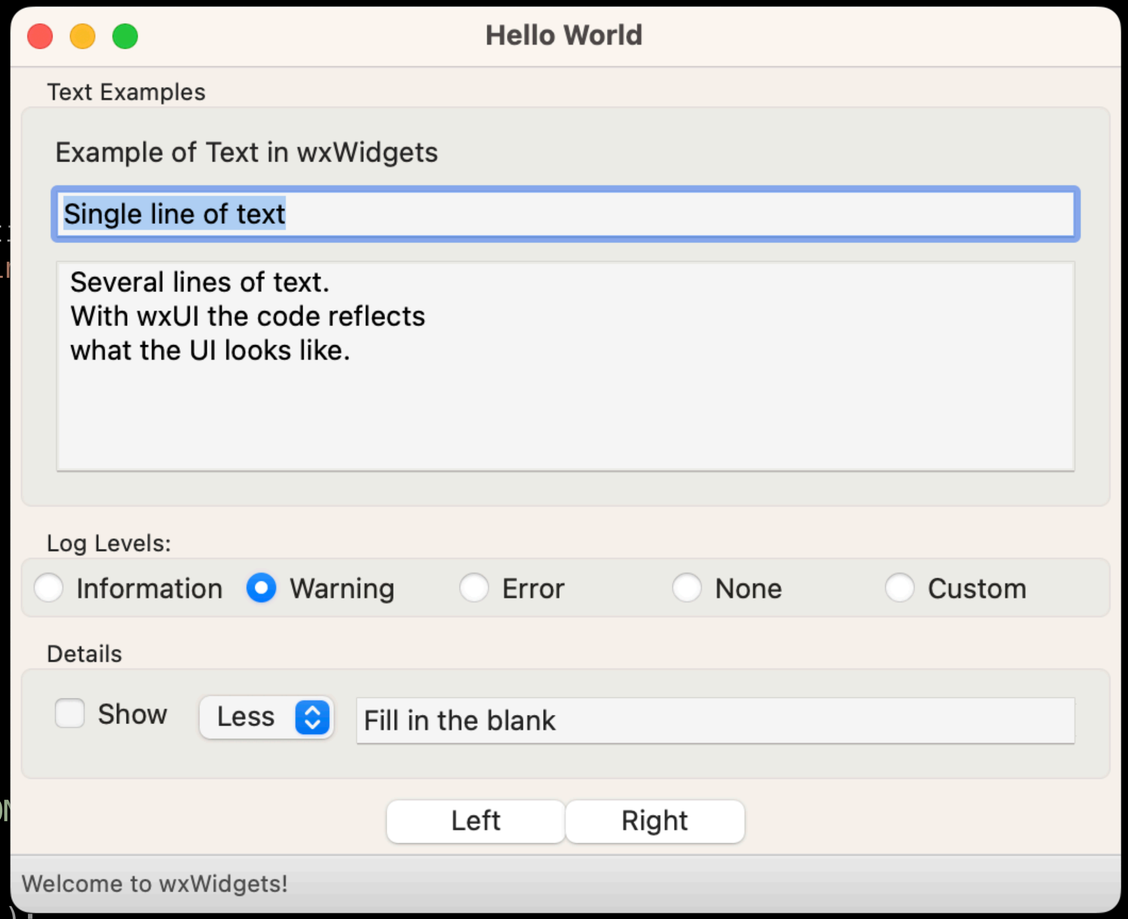
```
SetMenuBar(menuBar);

CreateStatusBar();
SetStatusText("Welcome to wxWidgets!");

// Create the controls.  This is cribbed from the RichTipDialog
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Example
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Single
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefau
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls
```



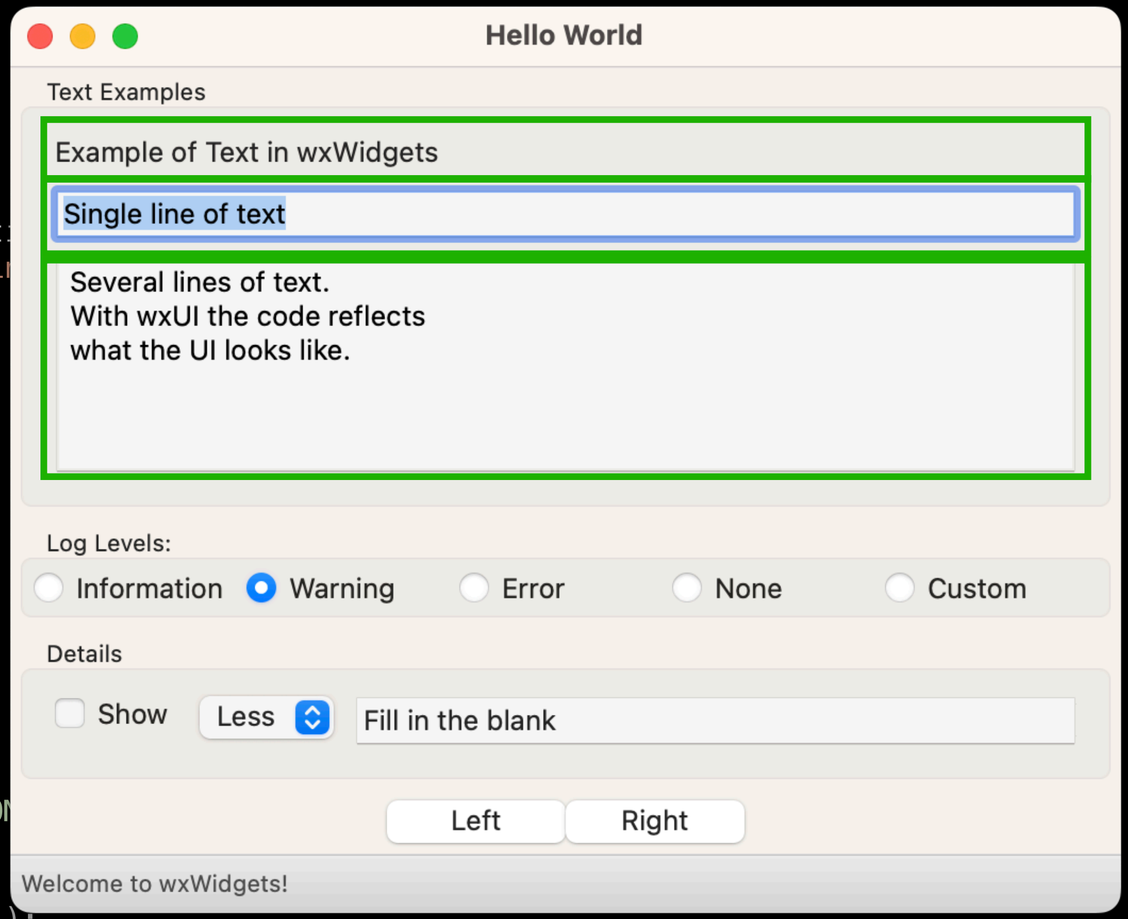
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Examples");
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "Details");
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosition, wxDefaultSize, WXSIZEOF(choices), choices);
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill in the blank");

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Examples");
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "Details");
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

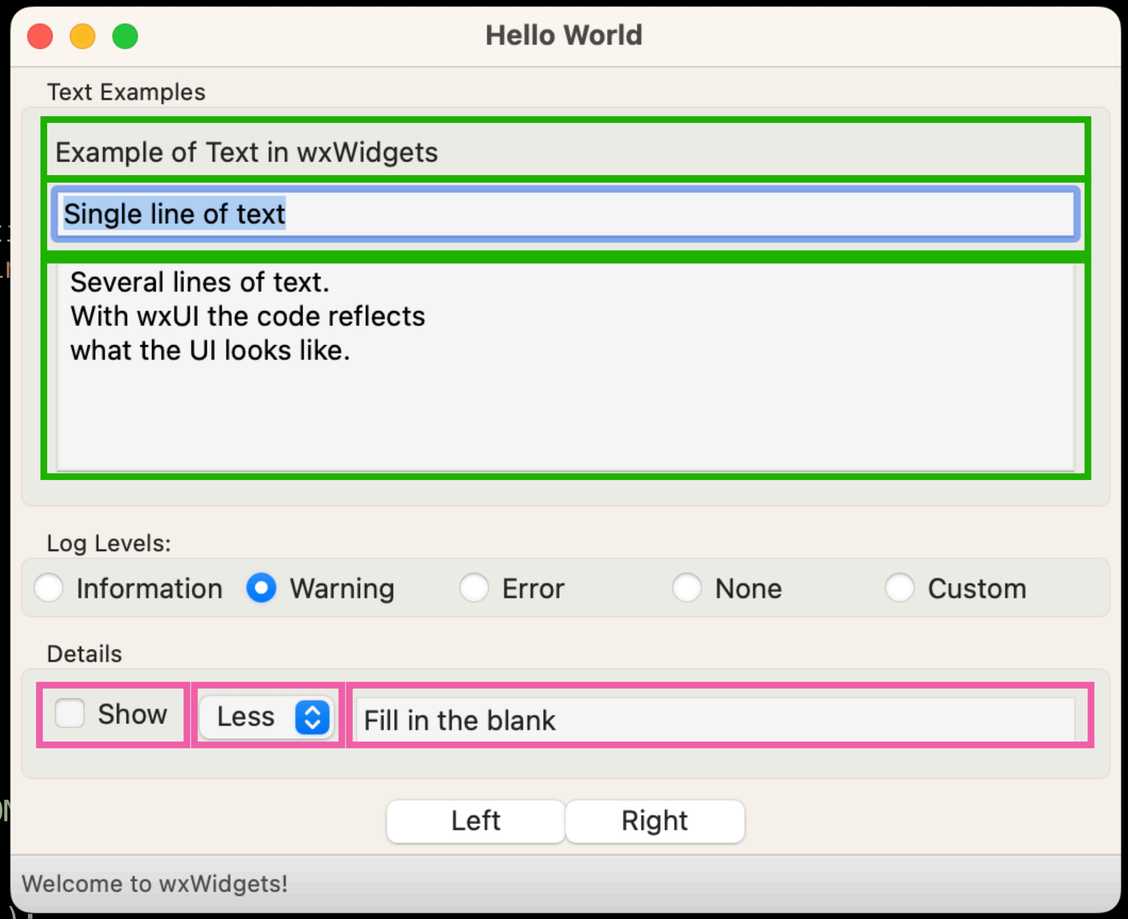
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPositi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill i

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



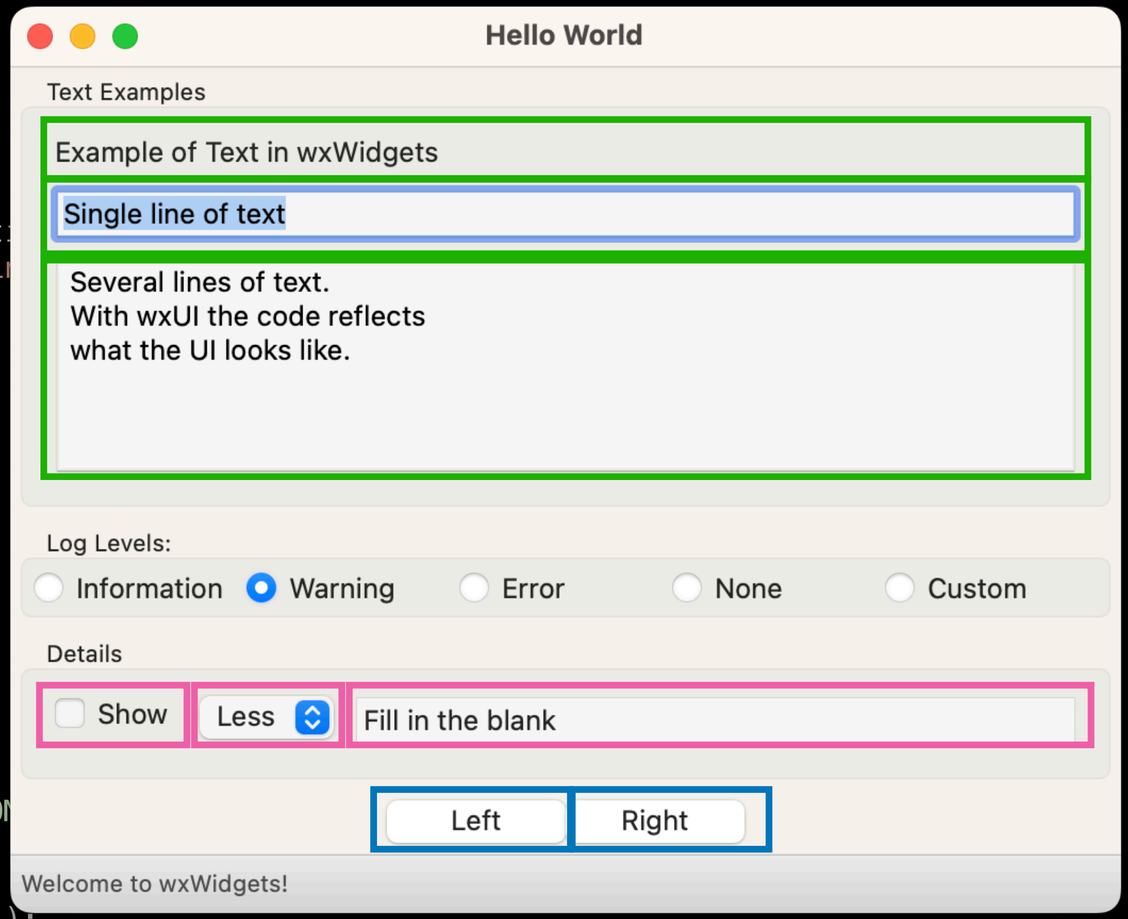
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPositi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill i

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



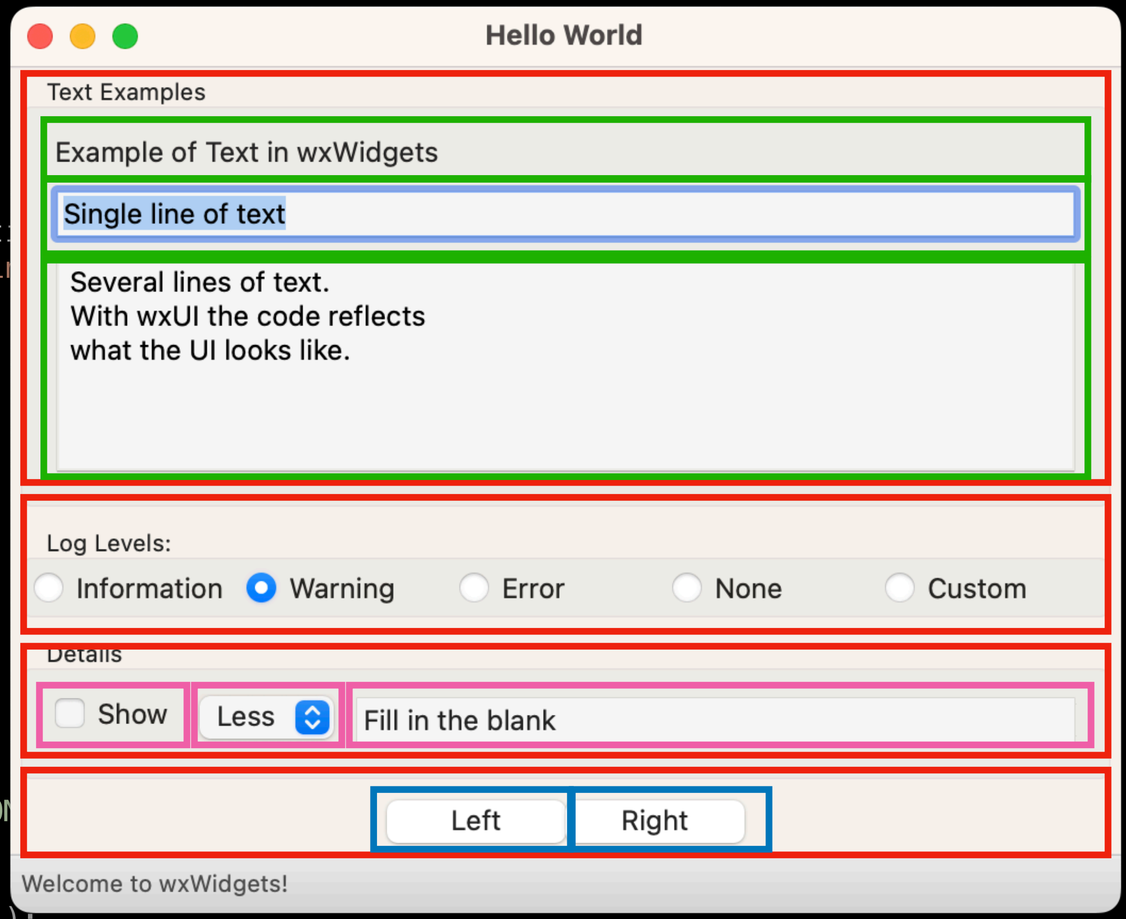
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPositi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill i

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



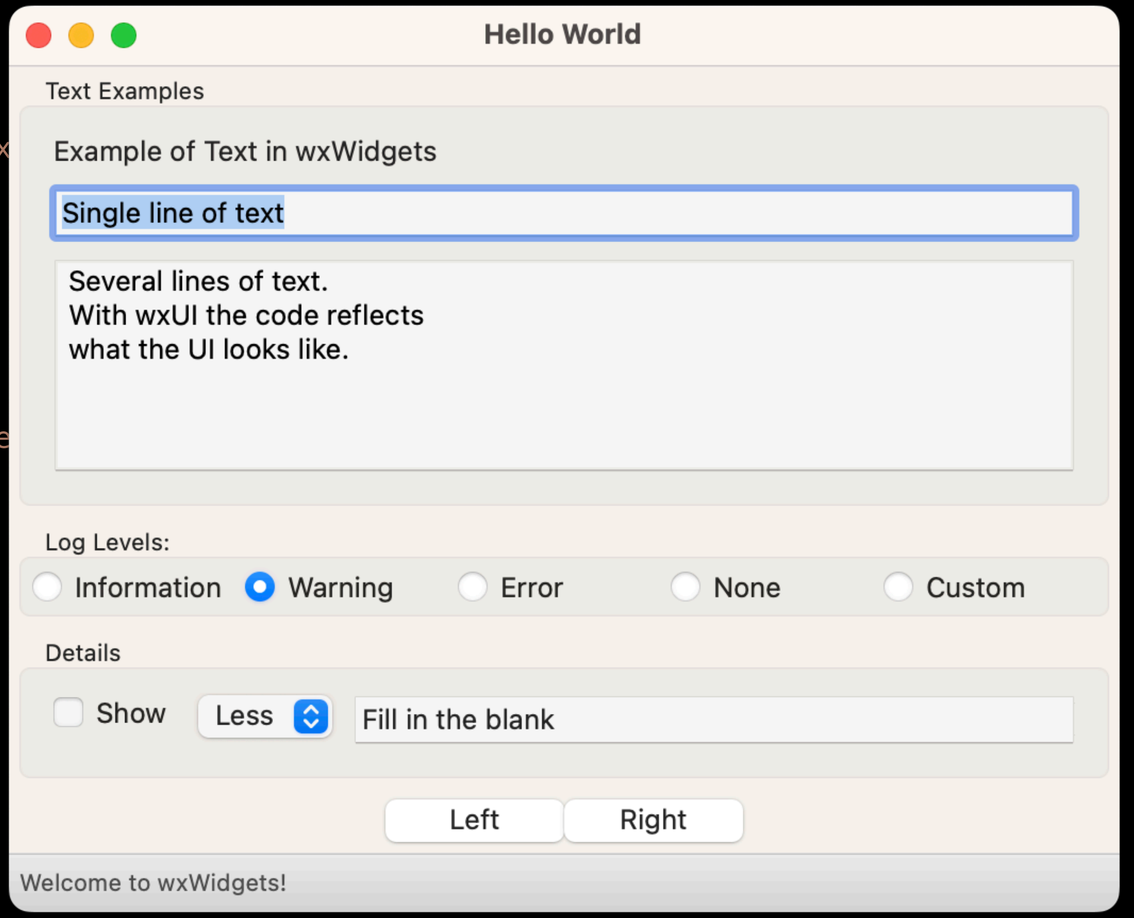
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPositi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill i

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



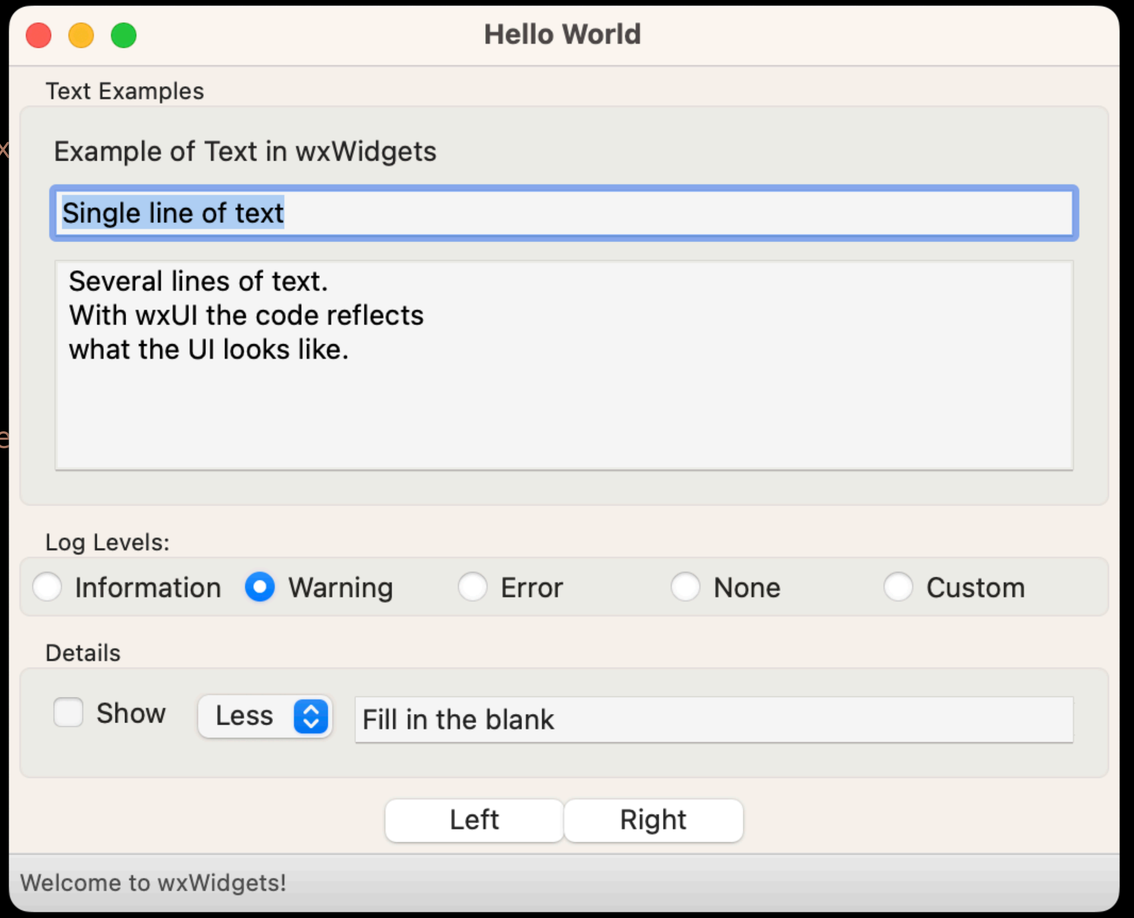
```
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefaultSize,
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show");
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPositi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill i

textBody->SetMinSize(wxSize(200, 100));

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



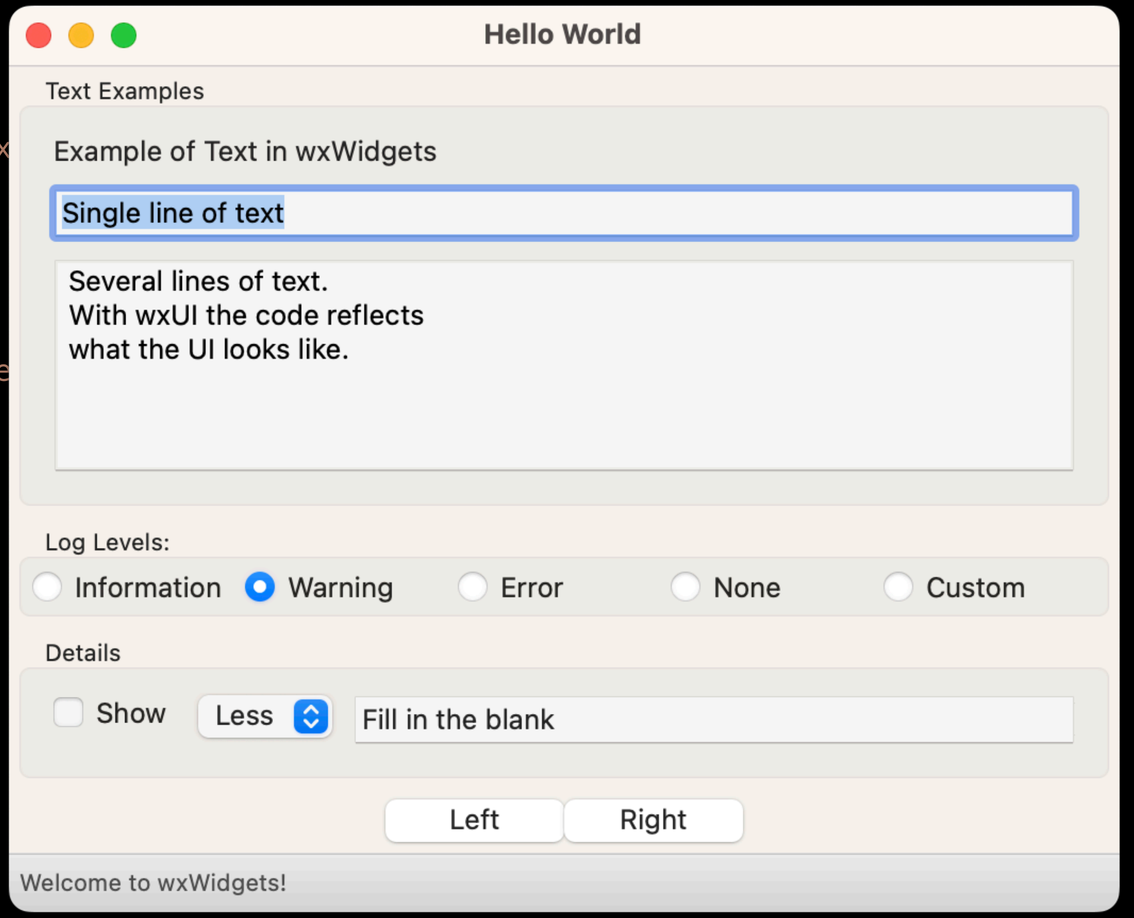
```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



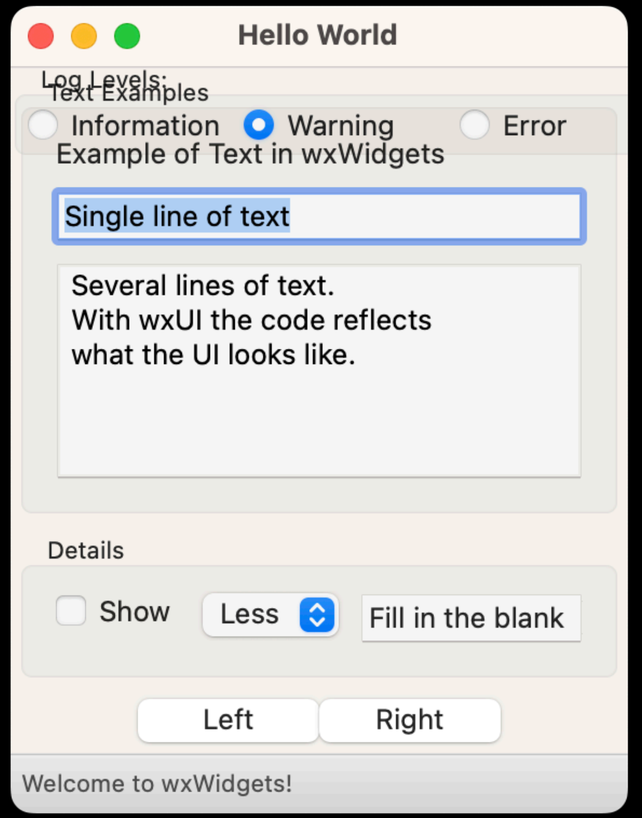
```cpp
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



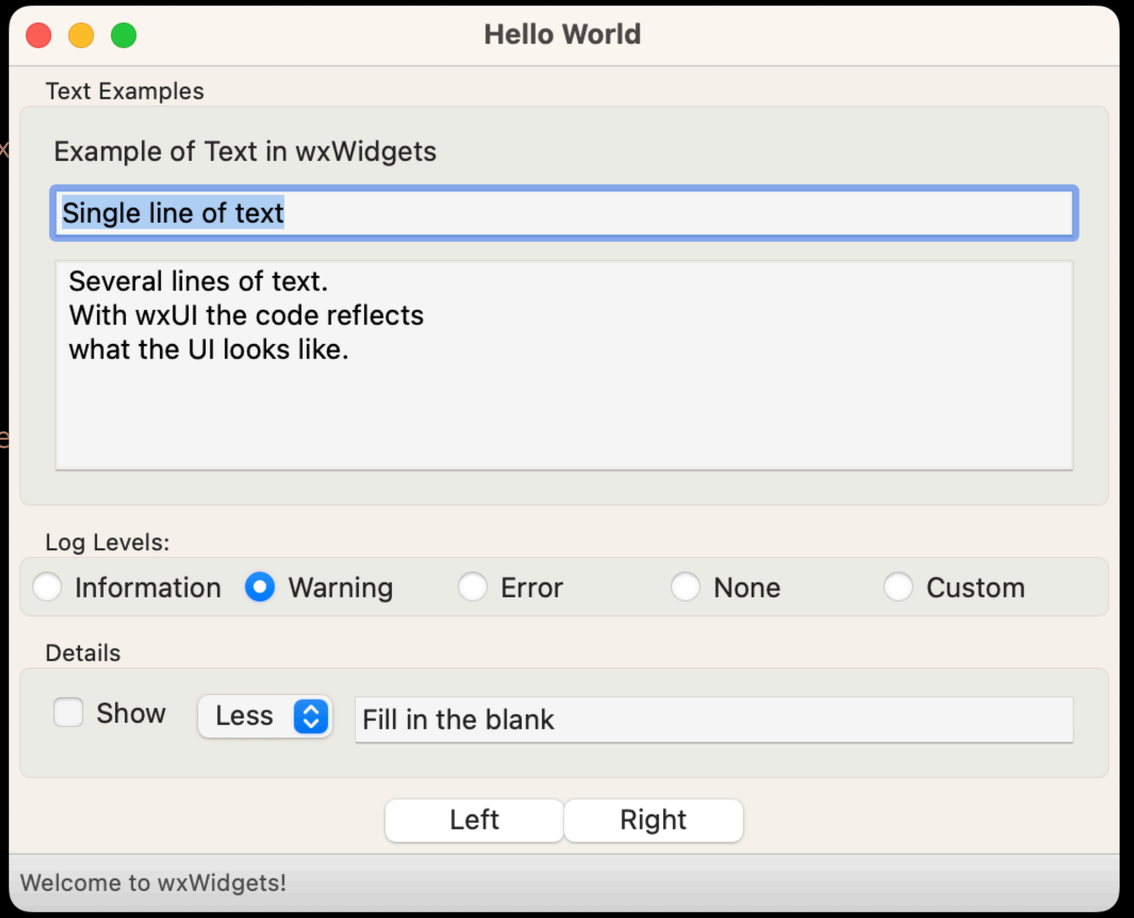
```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



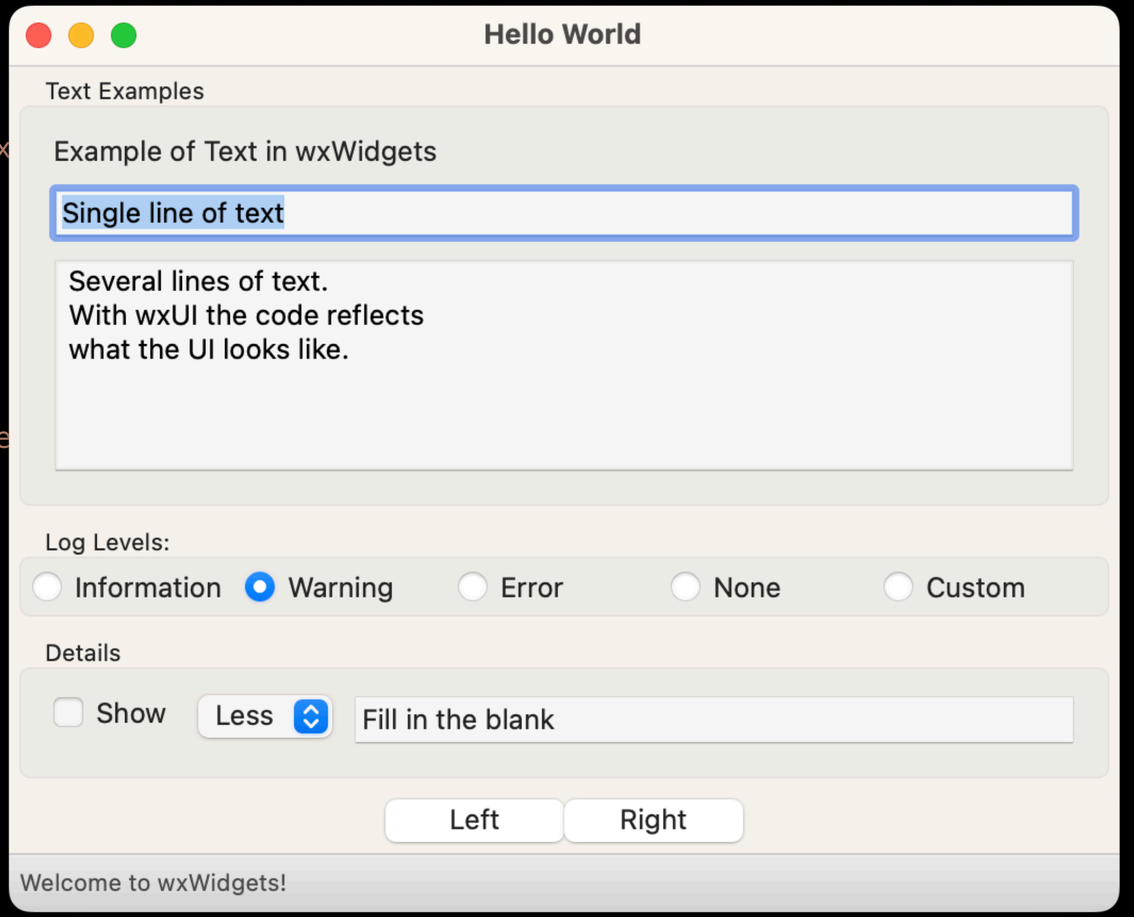
```cpp
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Examples");
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "Details");
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



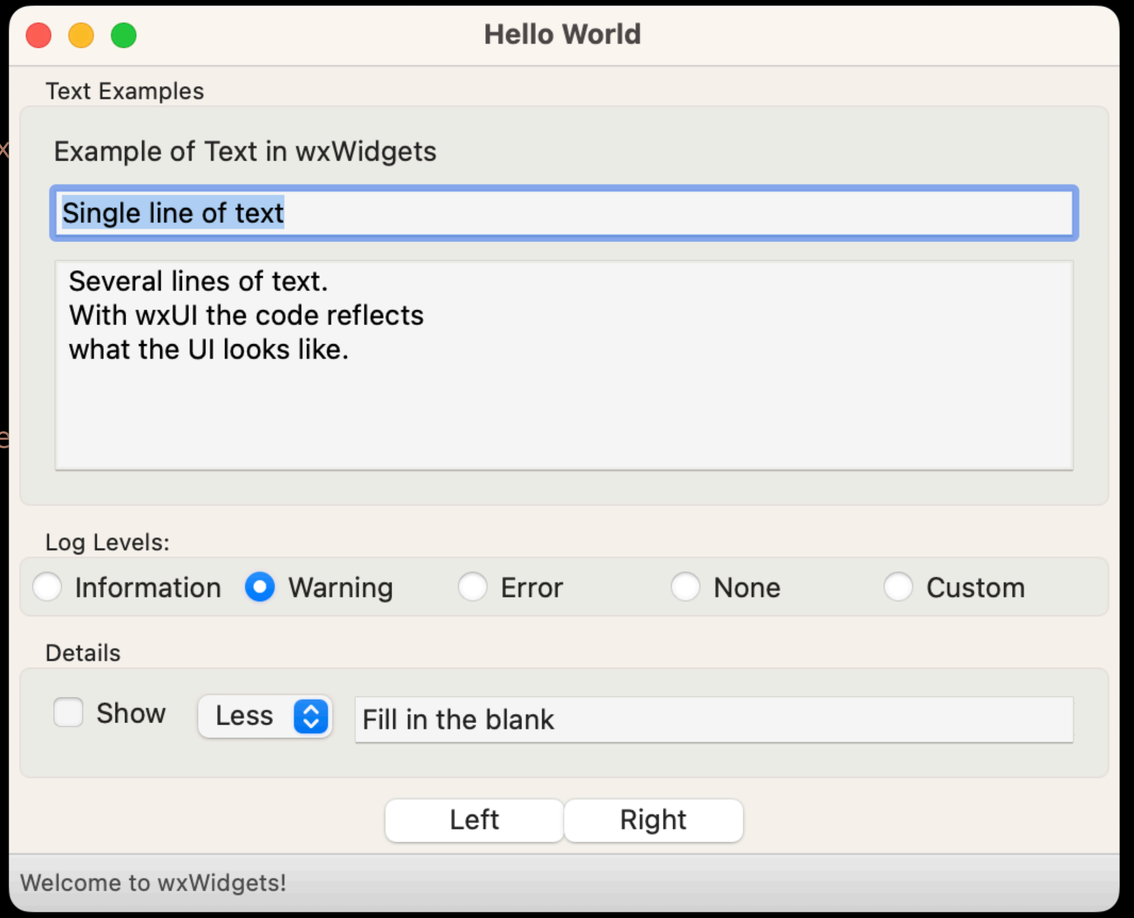
```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



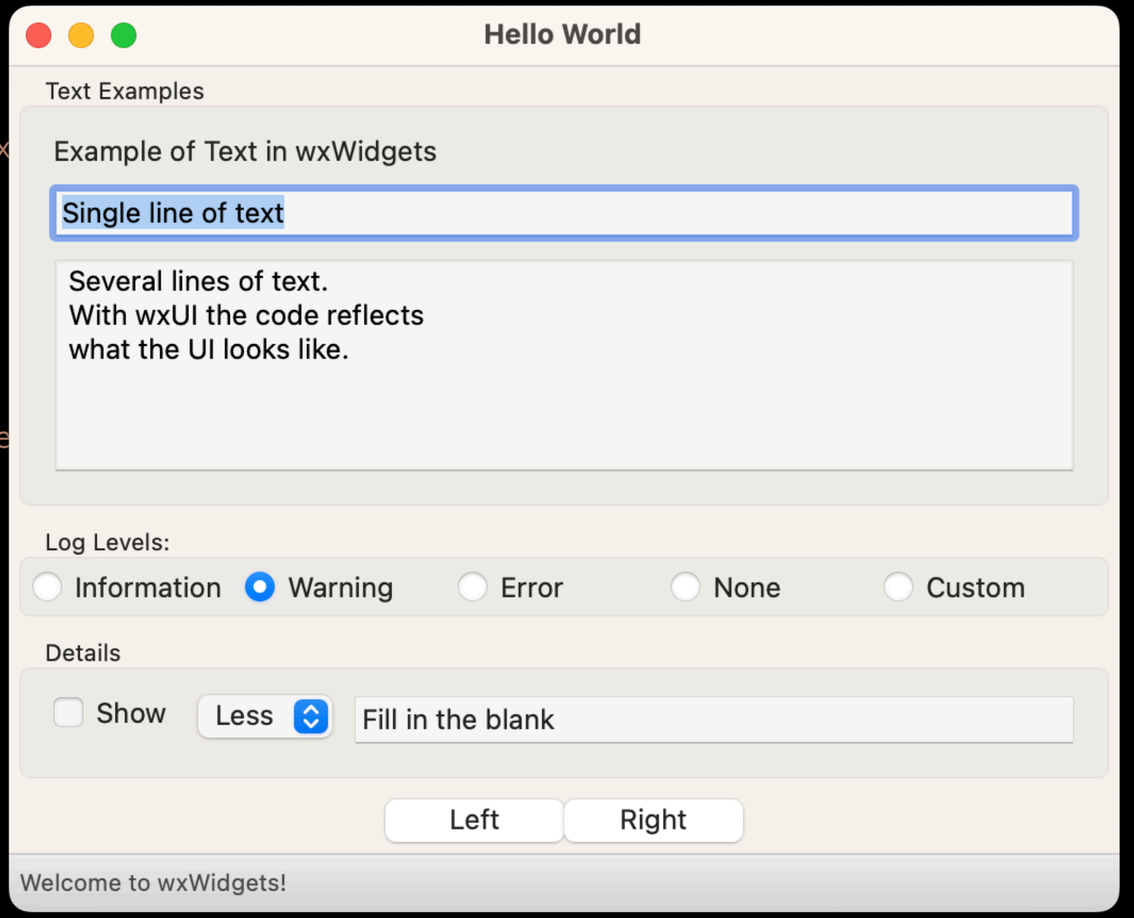
```cpp
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



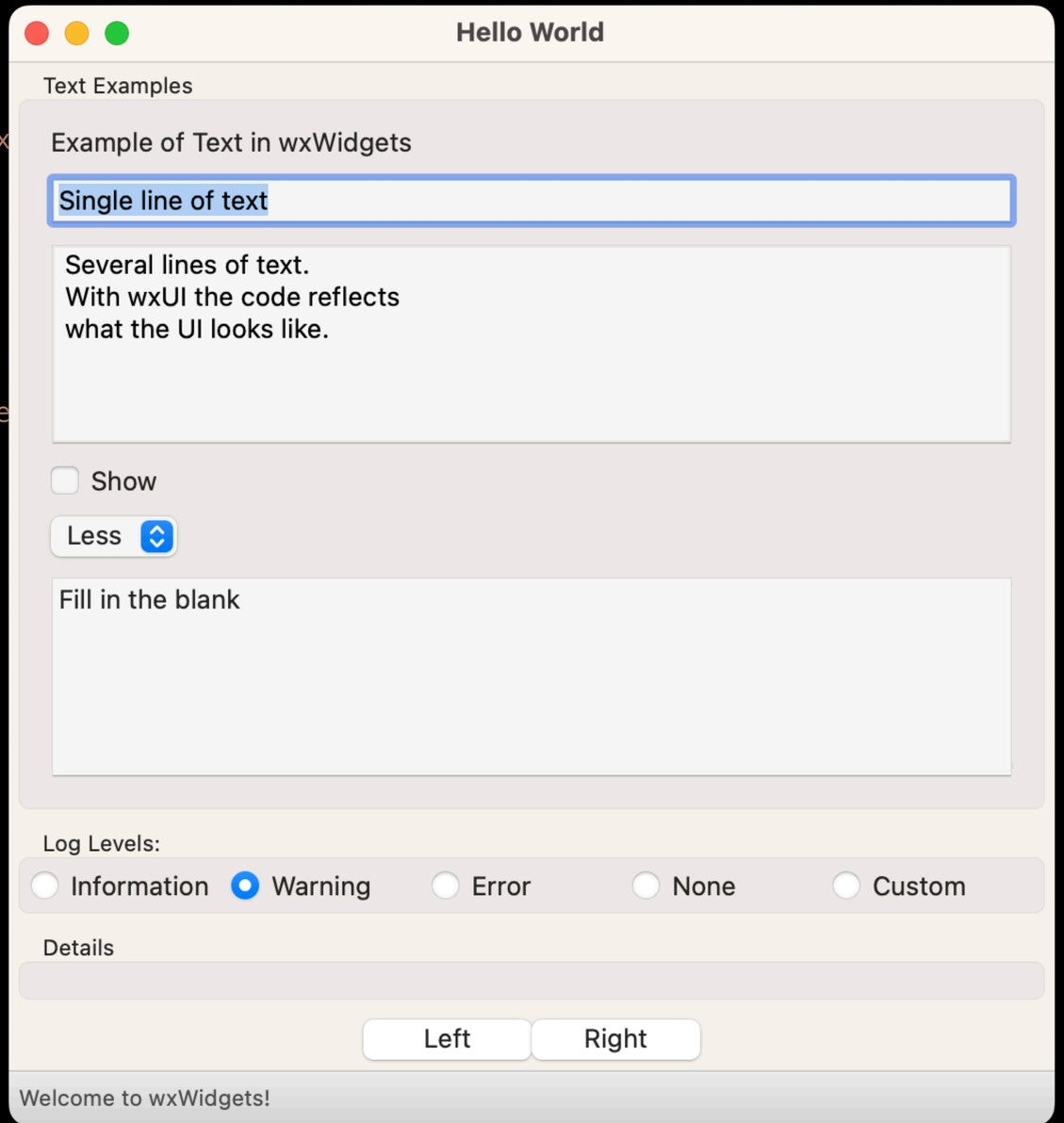
```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



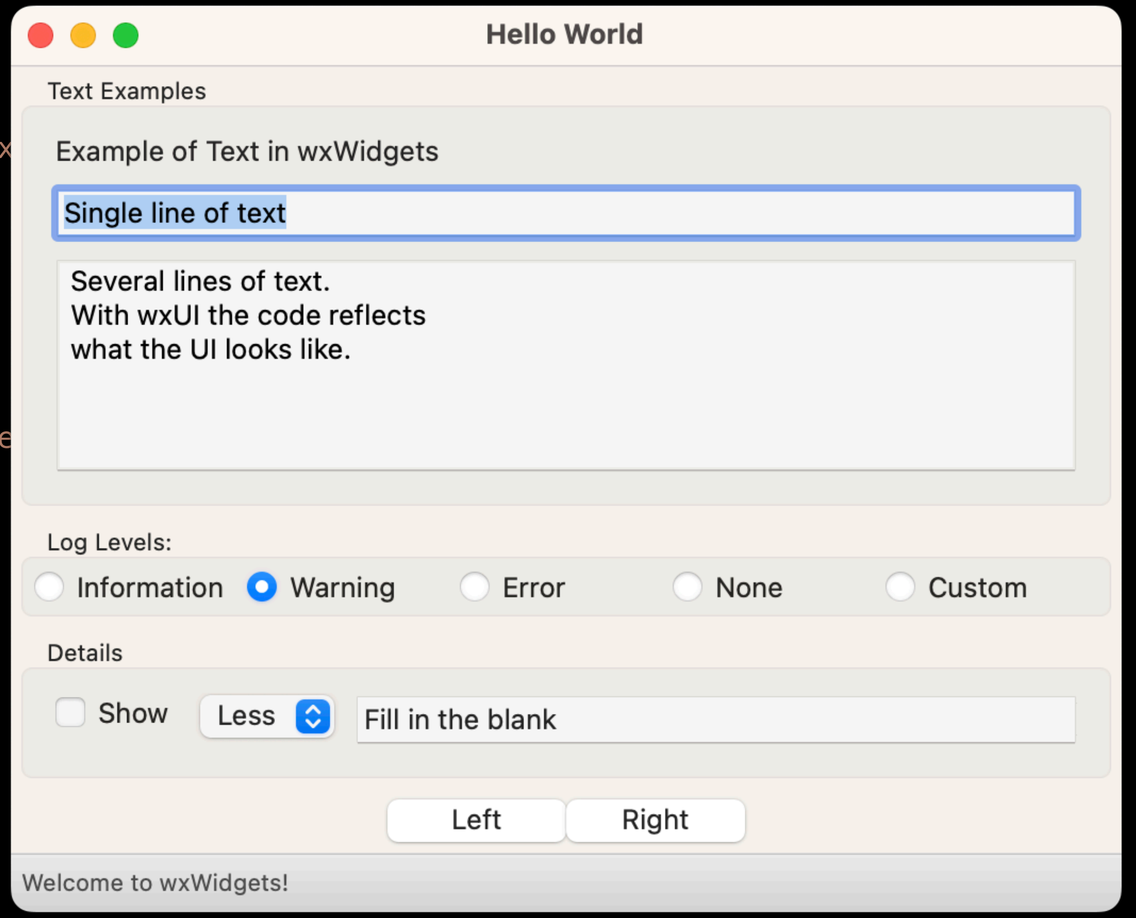
```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerText->Add(checkBox, wxSizerFlags().Border());
sizerText->Add(choice, wxSizerFlags().Border());
sizerText->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerText->Add(checkBox, wxSizerFlags().Border());
sizerText->Add(choice, wxSizerFlags().Border());
sizerText->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```



```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL, this, "Text Ex
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZONTAL, this, "De
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

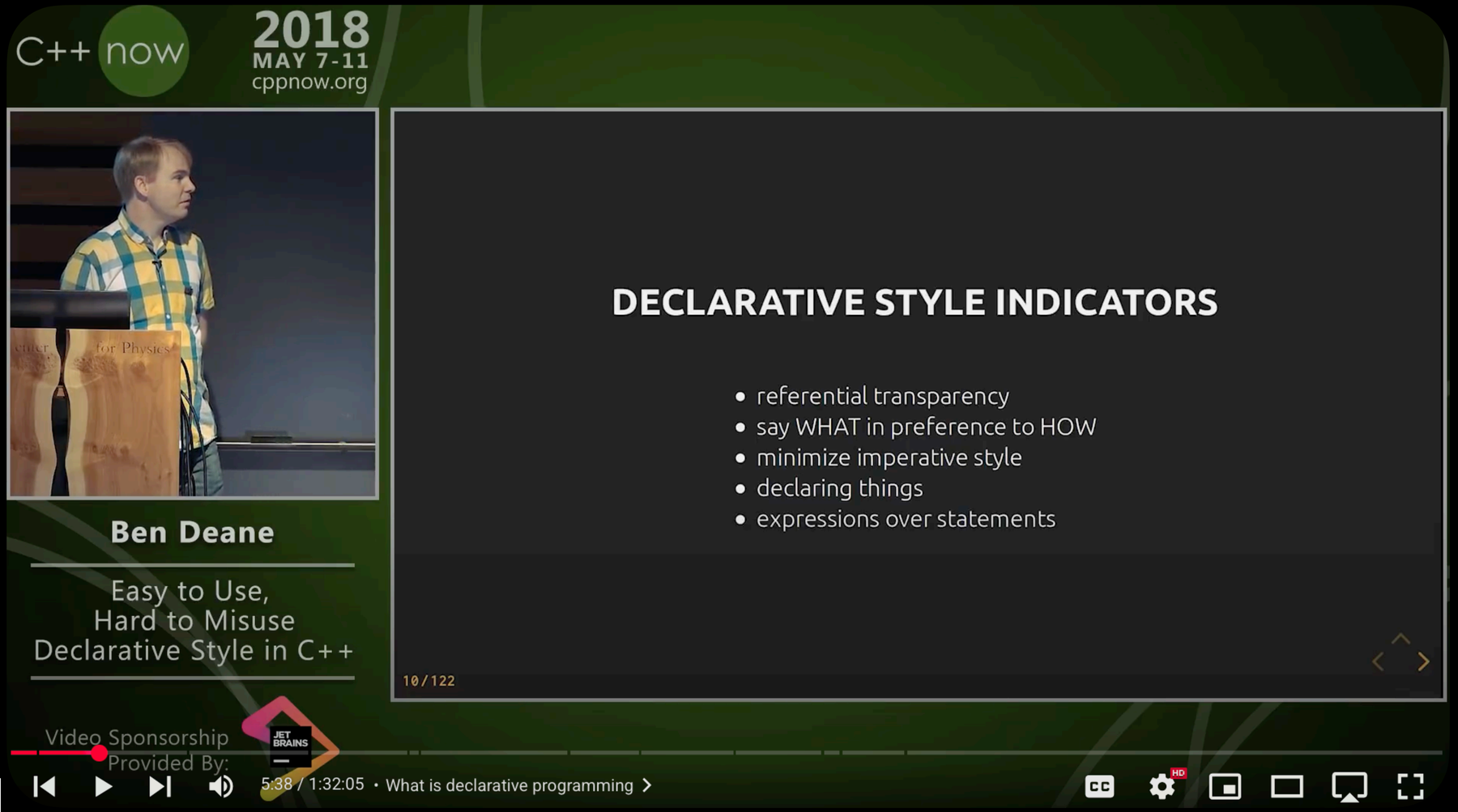


```
// Create the controls.  This is cribbed from the RichTipDial
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Exampl
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Singl
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefa
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show")
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill

textBody->SetMinSize(wxSize(200, 100));
```

```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border())
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

“Declarative programming is a non-imperative style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed.”

https://en.wikipedia.org/wiki/Declarative_programming

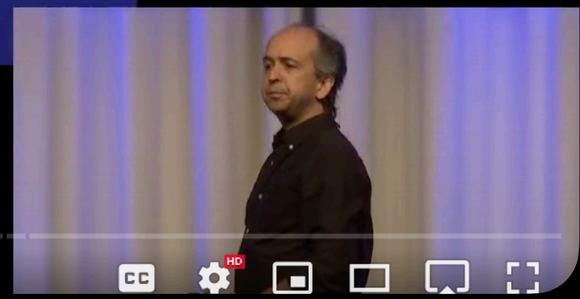


"[Declarative Programming] is preferring expressions over statements." - Ben Deane

https://youtu.be/2ouxETt75R4?si=OwM2-d07u5mTd2VI

Data Structures =
Programs

8:15 / 1:26:35 • Quest for meaning



Meeting C++ 2017
Kevlin Henney

Declarative thinking,
declarative practice

https://youtu.be/1s-BGBA8Nqo?si=X2n-7hTeZYqmNTuK

Declarative

```
// Create the controls.  This is cribbed from the RichTipDial
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Exampl
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Singl
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefa
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show")
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill

textBody->SetMinSize(wxSize(200, 100));
```

Imperative

```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border())
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

(more) Declarative

```
// Create the controls.  This is cribbed from the RichTipDial
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Exampl
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Singl
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefa
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show")
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill

textBody->SetMinSize(wxSize(200, 100));
```

(more) Imperative

```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border())
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

```
// Create the controls.  This is cribbed from the RichTipDial
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Exampl
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Singl
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefa
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show")
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill

textBody->SetMinSize(wxSize(200, 100));
```

```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border())
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

How can we make Layout more Declarative?

```
// Create the controls.  This is cribbed from the RichTipDial
wxStaticText* text = new wxStaticText(this, wxID_ANY, "Exampl
wxTextCtrl* textTitle = new wxTextCtrl(this, wxID_ANY, "Singl
wxTextCtrl* textBody = new wxTextCtrl(this, wxID_ANY,
    "Several lines of text.\n"
    "With wxUI the code reflects\n"
    "what the UI looks like.",
    wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

const wxString levels[] = {
    "&Information",
    "&Warning",
    "&Error",
    "&None",
    "&Custom",
};
wxRadioBox* logLevels = new wxRadioBox(
    this, wxID_ANY, "&Log Levels:", wxDefaultPosition, wxDefa
    WXSIZEOF(levels), levels, 1, wxRA_SPECIFY_ROWS);
logLevels->SetSelection(1);

wxCheckBox* checkBox = new wxCheckBox(this, wxID_ANY, "Show")
const wxString choices[] = {
    "Less",
    "More",
};

wxChoice* choice = new wxChoice(this, wxID_ANY, wxDefaultPosi
wxTextCtrl* blankLine = new wxTextCtrl(this, wxID_ANY, "Fill

textBody->SetMinSize(wxSize(200, 100));
```

```
// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxStaticBoxSizer* sizerText = new wxStaticBoxSizer(wxVERTICAL,
sizerText->Add(text, wxSizerFlags().Expand().Border());
sizerText->Add(textTitle, wxSizerFlags().Expand().Border());
sizerText->Add(textBody, wxSizerFlags(1).Expand().Border());
sizer->Add(sizerText, wxSizerFlags(1).Expand().Border());

sizer->Add(logLevels, wxSizerFlags().Expand().Border());

wxStaticBoxSizer* sizerDetails = new wxStaticBoxSizer(wxHORIZON
sizerDetails->Add(checkBox, wxSizerFlags().Border());
sizerDetails->Add(choice, wxSizerFlags().Border());
sizerDetails->Add(blankLine, wxSizerFlags(1).Expand().Border())
sizer->Add(sizerDetails, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Border(wxLEFT));
sizerBtns->Add(btnRight, wxSizerFlags().Border(wxRIGHT));
sizer->Add(sizerBtns, wxSizerFlags().Centre().Border());

SetSizerAndFit(sizer);
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Declare the Controls

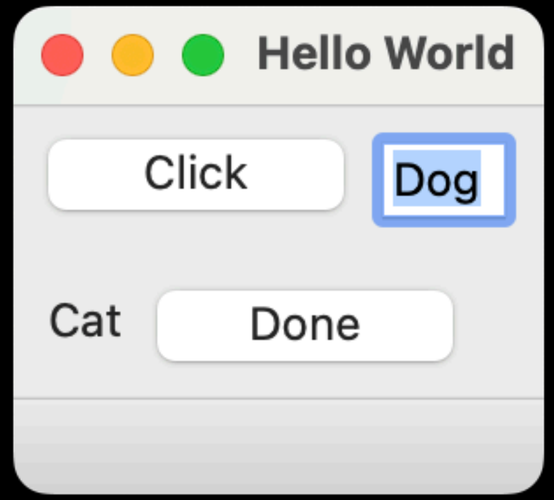
```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Layout the Controls

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Almost Always Auto

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```



Almost Always Auto

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    auto* button = new wxButton(this, wxID_ANY, "Click");
    auto* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    auto* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    auto* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    auto* button = new wxButton(this, wxID_ANY, "Click");
    auto* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    auto* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    auto* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    auto* button = new wxButton(this, wxID_ANY, "Click");
    auto* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    auto* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    auto* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    auto* button = new wxButton(this, wxID_ANY, "Click");
    auto* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    auto* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    auto* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

"Type" parameters

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

“Type” parameters

“Value” parameters

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
namespace DeclarativeUI {









}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

```
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

“Type” parameters

```cpp
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

Common

“Type” parameters

“Value” parameters

```
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(new wxButton(this, wxID_ANY, "Click"), wxSizerFlags().Border());
    sizerTop->Add(new wxTextCtrl(this, wxID_ANY, "Dog"), wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(new wxStaticText(this, wxID_ANY, "Cat"), wxSizerFlags().Border());
    sizerBottom->Add(new wxButton(this, wxID_EXIT, "Done"), wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
namespace DeclarativeUI {

template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    using namespace DeclarativeUI;
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}
```

Widget “Type”

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```cpp
template <typename Widget>
void CreateAndAdd(wxWindow* parent, wxWindowID id, std::string str, wxSizer* sizer, wxSizerFlags flags)
{
    sizer->Add(new Widget(parent, id, str), flags);
}
```

Widget “Type”

Widget “state”

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
template <typename W>
struct Widget {









};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

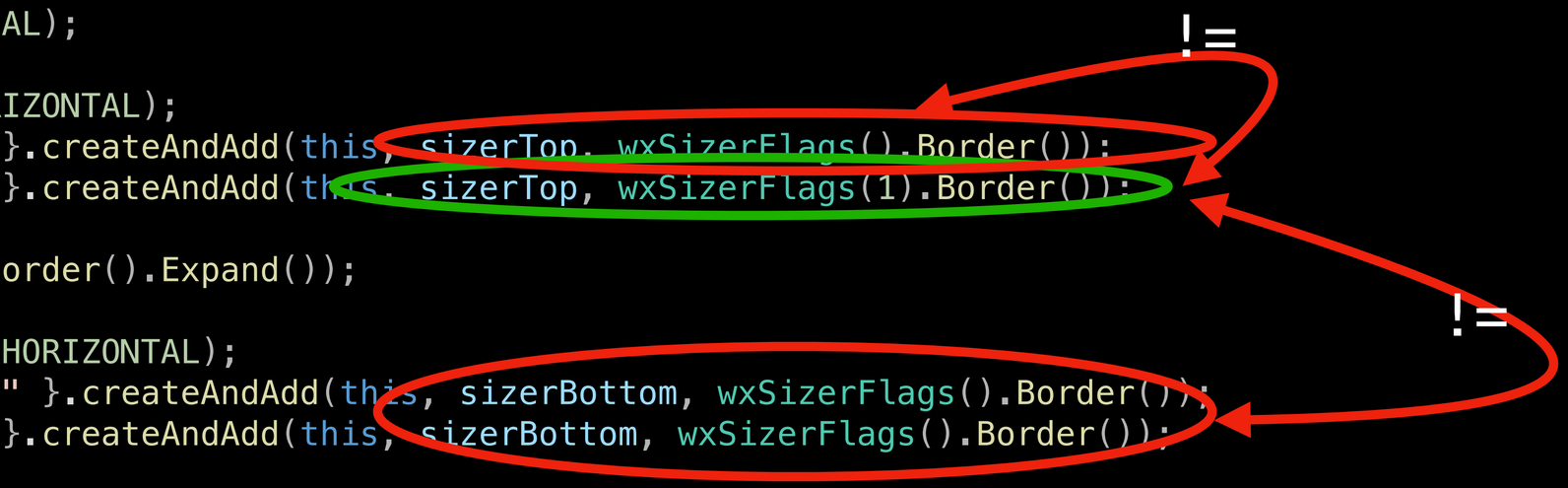
```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }




private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;

};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags)
    {
        sizer->Add(new W(parent, id_, str_), flags);
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;

};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxButton>(this, wxID_ANY, "Click", sizerTop, wxSizerFlags().Border());
    CreateAndAdd<wxTextCtrl>(this, wxID_ANY, "Dog", sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    CreateAndAdd<wxStaticText>(this, wxID_ANY, "Cat", sizerBottom, wxSizerFlags().Border());
    CreateAndAdd<wxButton>(this, wxID_EXIT, "Done", sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags)
    {
        sizer->Add(new W(parent, id_, str_), flags);
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;

};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog" }.createAndAdd(this, sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags)
    {
        sizer->Add(new W(parent, id_, str_), flags);
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;

};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog" }.createAndAdd(this, sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags)
    {
        sizer->Add(new W(parent, id_, str_), flags);
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;

};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog" }.createAndAdd(this, sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```



```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }
    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog" }.createAndAdd(this, sizerTop, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }
    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog", wxSizerFlags(1).Border() }.createAndAdd(this, sizerTop, wxSizerFlags().Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }
    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }
private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog", wxSizerFlags(1).Border() }.createAndAdd(this, sizerTop, wxSizerFlags().Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    // Create and layout the controls.
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxButton> { wxID_ANY, "Click" }.createAndAdd(this, sizerTop, wxSizerFlags().Border());
    Widget<wxTextCtrl> { wxID_ANY, "Dog", wxSizerFlags(1).Border() }.createAndAdd(this, sizerTop, wxSizerFlags().Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
    Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);
```

```
auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" } createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }   createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" } createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }   createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

Heterogeneous Collection

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" } createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }   createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

Heterogeneous Collection

`std::tuple`

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

How to call a function on each member?

std::apply

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

How to call a function on each member?

std::apply

Defined in header <tuple>

```
template< class F, class Tuple >
constexpr decltype(auto) apply( F&& f, Tuple&& t );                          (since C++17) (until C++23)

template< class F, class Tuple >
constexpr decltype(auto) apply( F&& f, Tuple&& t ) noexcept(/* see below */); (since C++23)
```

Invoke the *Callable* object f with a tuple of arguments.

```cpp
template<typename... Ts>
std::ostream& operator<<(std::ostream& os, std::tuple<Ts...> const& theTuple)
{
    std::apply
    (
        [&os](Ts const&... tupleArgs)
        {
            os << '[';
            std::size_t n{0};
            ((os << tupleArgs << (++n != sizeof...(Ts) ? ", " : "")), ...);
            os << ']';
        }, theTuple
    );
    return os;
}
```

https://en.cppreference.com/w/cpp/utility/apply

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
Widget<wxStaticText> { wxID_ANY, "Cat" }.createAndAdd(this, sizerBottom, wxSizerFlags().Border());
Widget<wxButton> { wxID_EXIT, "Done" }  .createAndAdd(this, sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
auto bottomWidgets = std::tuple {
    Widget<wxStaticText> { wxID_ANY, "Cat" },
    Widget<wxButton> { wxID_EXIT, "Done" },
};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
auto bottomWidgets = std::tuple {
    Widget<wxStaticText> { wxID_ANY, "Cat" },
    Widget<wxButton> { wxID_EXIT, "Done" },
};
std::apply([this, sizer = sizerBottom](auto&&... tupleArg) {
    (tupleArg.createAndAdd(this, sizer, wxSizerFlags().Border()), ...);
},
    bottomWidgets);
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
auto bottomWidgets = std::tuple {
    Widget<wxStaticText> { wxID_ANY, "Cat" },
    Widget<wxButton> { wxID_EXIT, "Done" },
};
std::apply([this, sizer = sizerBottom, flags = wxSizerFlags().Border()](auto&&... tupleArg) {
    (tupleArg.createAndAdd(this, sizer, flags), ...);
},
    bottomWidgets);
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
auto bottomWidgets = std::tuple {
    Widget<wxStaticText> { wxID_ANY, "Cat" },
    Widget<wxButton> { wxID_EXIT, "Done" },
};
std::apply([this, sizer = sizerBottom, flags = wxSizerFlags().Border()](auto&&... tupleArg) {
    (tupleArg.createAndAdd(this, sizer, flags), ...);
},
    bottomWidgets);
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    auto bottomWidgets = std::tuple {
        Widget<wxStaticText> { wxID_ANY, "Cat" },
        Widget<wxButton> { wxID_EXIT, "Done" },
    };
    std::apply([this, sizer = sizerBottom, flags = wxSizerFlags().Border()](auto&&... tupleArg) {
        (tupleArg.createAndAdd(this, sizer, flags), ...);
    },
        bottomWidgets);
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    auto bottomWidgets = std::tuple {
        Widget<wxStaticText> { wxID_ANY, "Cat" },
        Widget<wxButton> { wxID_EXIT, "Done" },
    };
    std::apply([this, sizer = sizerBottom, flags = wxSizerFlags().Border()](auto&&... tupleArg) {
        (tupleArg.createAndAdd(this, sizer, flags), ...);
    },
        bottomWidgets);
```

```
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}


    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        std::tuple {
            Widget<wxStaticText> { wxID_ANY, "Cat" },
            Widget<wxButton> { wxID_EXIT, "Done" },
        });
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        std::tuple {
            Widget<wxStaticText> { wxID_ANY, "Cat" },
            Widget<wxButton> { wxID_EXIT, "Done" },
        });
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Widget<wxStaticText> { wxID_ANY, "Cat" },
        Widget<wxButton> { wxID_EXIT, "Done" });
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}
```

```cpp
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Widget<wxStaticText> { wxID_ANY, "Cat" },
        Widget<wxButton> { wxID_EXIT, "Done" });
```

```cpp
namespace DeclarativeUI {

using TextCtrl = Widget<wxTextCtrl>;
using Button = Widget<wxButton>;
using Text = Widget<wxStaticText>;

}
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}
```

```cpp
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```cpp
namespace DeclarativeUI {

using TextCtrl = Widget<wxTextCtrl>;
using Button = Widget<wxButton>;
using Text = Widget<wxStaticText>;

}
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```cpp
template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}



    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```
template <typename T>
concept CreateAndAddable = requires(T widget, wxWindow* window, wxSizer* sizer) {
    widget.createAndAdd(window, sizer, wxSizerFlags {});
};

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <typename... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}


    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```
template <typename T>
concept CreateAndAddable = requires(T widget, wxWindow* window, wxSizer* sizer) {
    widget.createAndAdd(window, sizer, wxSizerFlags {});
};

template <CreateAndAddable... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <CreateAndAddable... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}


    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```cpp
template <typename T>
concept CreateAndAddable = requires(T widget, wxWindow* window, wxSizer* sizer) {
    widget.createAndAdd(window, sizer, wxSizerFlags {});
};

template <CreateAndAddable... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, std::tuple<W...> widgets)
{
    std::apply([parent, sizer, flags](auto&&... tupleArg) {
        (tupleArg.createAndAdd(parent, sizer, flags), ...);
    },
        widgets);
}

template <CreateAndAddable... W>
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags flags, W... widgets)
{
    return createAndAdd(parent, sizer, flags, std::make_tuple(widgets...));
}
```

```cpp
    auto* sizer = new wxBoxSizer(wxVERTICAL);

    auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    createAndAdd(this,
        sizerBottom,
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" });
```

```cpp
namespace DeclarativeUI {

static_assert(CreateAndAddable<TextCtrl>);
static_assert(CreateAndAddable<Button>);
static_assert(CreateAndAddable<Text>);

}
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });
```

```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

Common

```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

Common

“Type” parameters

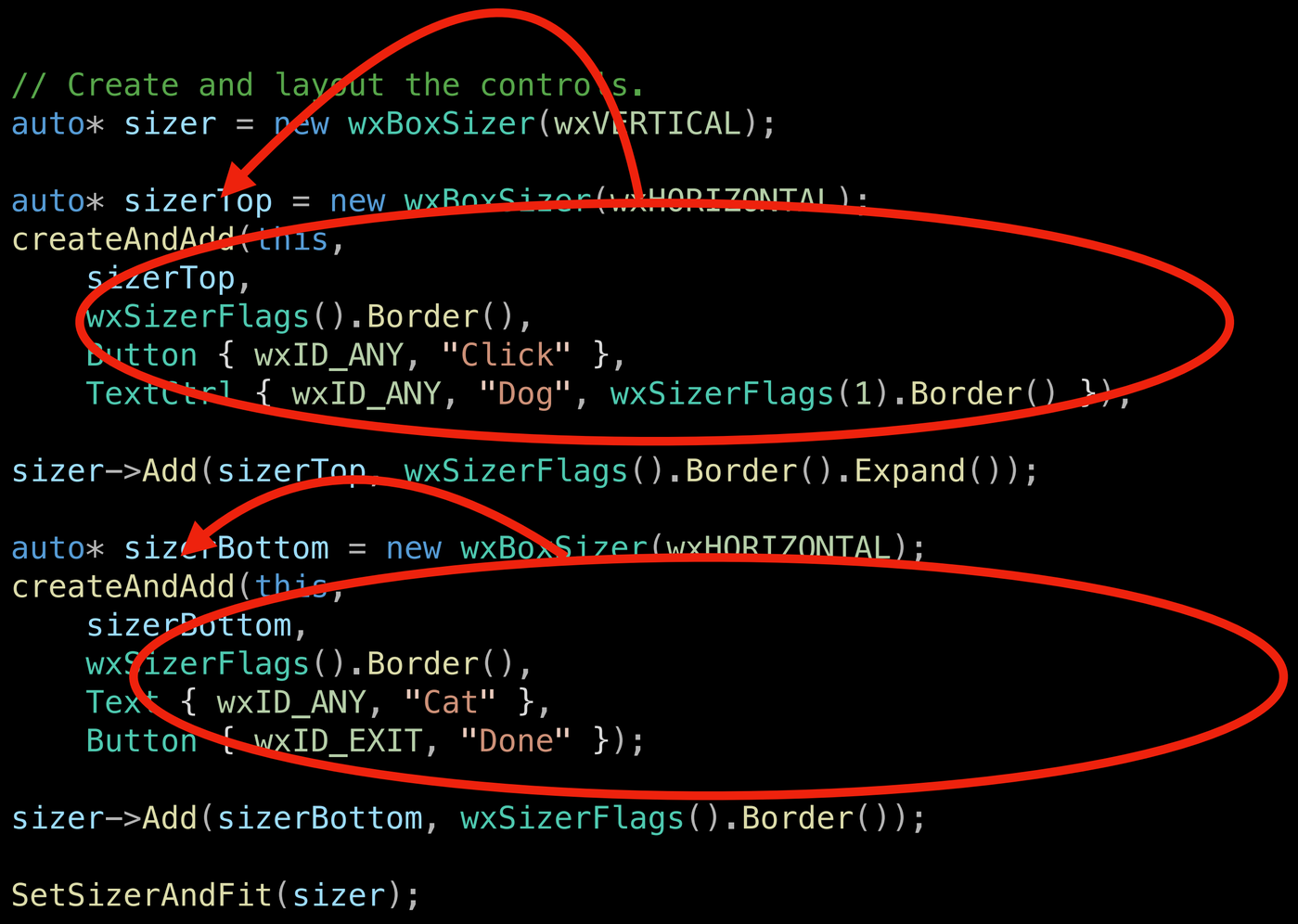
```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

Common

“Type” parameters

“Value” parameters

```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```



```
// Create and layout the controls.
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerTop = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerTop,
    wxSizerFlags().Border(),
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() });

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

Constructs

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        return sizer;
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
createAndAdd(this,
    sizerBottom,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" });

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        return sizer;
    }


    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this);

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this);

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this);

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        return sizer;
    }


    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this);

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

auto* sizerBottom = Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this);

sizer->Add(sizerBottom, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    wxSizerFlags flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    std::optional<wxSizerFlags> flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    wxSizerFlags().Border(),
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    std::optional<wxSizerFlags> flags;
    std::tuple<W...> widgets;
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    std::optional<wxSizerFlags> flags;
    std::tuple<W...> widgets;
};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer;

template <CreateAndAddable... W>
struct HSizer : Sizer<W...> {
    HSizer(W... widgets)
        : Sizer<W...>(wxHORIZONTAL, widgets...)
    {
    }
    HSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxHORIZONTAL, flags, widgets...)
    {
    }
};

template <CreateAndAddable... W>
struct VSizer : Sizer<W...> {
    VSizer(W... widgets)
        : Sizer<W...>(wxVERTICAL, widgets...)
    {
    }
    VSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxVERTICAL, flags, widgets...)
    {
    }
};
```

```cpp
template <CreateAndAddable... W>
struct Sizer;

template <CreateAndAddable... W>
struct HSizer : Sizer<W...> {
    HSizer(W... widgets)
        : Sizer<W...>(wxHORIZONTAL, widgets...)
    {
    }
    HSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxHORIZONTAL, flags, widgets...)
    {
    }
};

template <CreateAndAddable... W>
struct VSizer : Sizer<W...> {
    VSizer(W... widgets)
        : Sizer<W...>(wxVERTICAL, widgets...)
    {
    }
    VSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxVERTICAL, flags, widgets...)
    {
    }
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

Sizer {
    wxHORIZONTAL,
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```cpp
template <CreateAndAddable... W>
struct Sizer;

template <CreateAndAddable... W>
struct HSizer : Sizer<W...> {
    HSizer(W... widgets)
        : Sizer<W...>(wxHORIZONTAL, widgets...)
    {
    }
    HSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxHORIZONTAL, flags, widgets...)
    {
    }
};

template <CreateAndAddable... W>
struct VSizer : Sizer<W...> {
    VSizer(W... widgets)
        : Sizer<W...>(wxVERTICAL, widgets...)
    {
    }
    VSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxVERTICAL, flags, widgets...)
    {
    }
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    std::optional<wxSizerFlags> flags;
    std::tuple<W...> widgets;
};
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

    wxOrientation orientation;
    std::optional<wxSizerFlags> flags;
    std::tuple<W...> widgets;
};
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {

    }

    wx
    st
    st
};
```

```cpp
template <typename T>
concept CreateAndAddable
    = requires(T widget, wxWindow* window, wxSizer* sizer) {
        widget.createAndAdd(window, sizer, wxSizerFlags {});
    };
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {

    }

    wx
    st
    st
};
```

```
template <typename T>
concept CreateAndAddable
    = requires(T widget, wxWindow* window, wxSizer* sizer) {
        widget.createAndAdd(window, sizer, wxSizerFlags {});
    };
```

```
static_assert(CreateAndAddable<HSizer<Button, TextCtrl>>);
static_assert(CreateAndAddable<HSizer<Text, Button>>);
```

```
template <CreateAndAddable... W>
struct Sizer {
    Sizer(wxOrientation orientation, wxSizerFlags flags, W... widgets)
        : orientation(orientation)
        , flags(flags)
        , widgets(std::make_tuple(widgets...))
    {
    }
    Sizer(wxOrientation orientation, W... widgets)
        : orientation(orientation)
        , widgets(std::make_tuple(widgets...))
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {

    }

    wx
    st
    st
};
```

```
static_assert(CreateAndAddable<HSizer<Button, TextCtrl>>);
static_assert(CreateAndAddable<HSizer<Text, Button>>);

static_assert(CreateAndAddable<VSizer<HSizer<Button, TextCtrl>,
                                      HSizer<Text, Button>>>);
```

```
template <typename T>
concept CreateAndAddable
    = requires(T widget, wxWindow* window, wxSizer* sizer) {
          widget.createAndAdd(window, sizer, wxSizerFlags {});
      };
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() }
}.createAndAdd(this, sizer, wxSizerFlags().Border().Expand());

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() }
}.createAndAdd(this, sizer, wxSizerFlags().Border().Expand());

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        parentSizer->Add(sizer, parentFlags);
        return sizer;
    }

};
```

```
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() }
}.createAndAdd(this, sizer, wxSizerFlags().Border().Expand());

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() }
}.createAndAdd(this, sizer, wxSizerFlags().Border().Expand());

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```cpp
auto* sizer = new wxBoxSizer(wxVERTICAL);

HSizer {
    Button { wxID_ANY, "Click" },
    TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border() }
}.createAndAdd(this, sizer, wxSizerFlags().Border().Expand());

HSizer {
    Text { wxID_ANY, "Cat" },
    Button { wxID_EXIT, "Done" }
}.createAndAdd(this, sizer, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```cpp
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand())
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }

};
```

```
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }

};
```

```cpp
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```cpp
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }
};
```

```cpp
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

this->SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }

    auto fitTo(wxWindow* parent)
    {
        auto* sizer = createAndAdd(parent,
              flags.value_or(wxSizerFlags()));
        parent->SetSizerAndFit(sizer);
        return sizer;
    }
};
```

```cpp
auto* sizer = VSizer {
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.createAndAdd(this, wxSizerFlags().Border());

this->SetSizerAndFit(sizer);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }

    auto fitTo(wxWindow* parent)
    {
        auto* sizer = createAndAdd(parent,
              flags.value_or(wxSizerFlags()));
        parent->SetSizerAndFit(sizer);
        return sizer;
    }
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
template <CreateAndAddable... W>
struct Sizer {
    auto createAndAdd(wxWindow* parent, wxSizerFlags parentFlags)
    {
        auto* sizer = new wxBoxSizer(orientation);
        std::apply([this, parent, sizer, parentFlags](auto&&... tupleArg) {
            (tupleArg.createAndAdd(parent, sizer, flags.value_or(parentFlags)), ...);
        },
            widgets);
        return sizer;
    }

    auto createAndAdd(wxWindow* parent, wxSizer* parentSizer, wxSizerFlags parentFlags)
    {
        auto* sizer = createAndAdd(parent, flags.value_or(parentFlags));
        parentSizer->Add(sizer, flags.value_or(parentFlags));
        return sizer;
    }

    auto fitTo(wxWindow* parent)
    {
        auto* sizer = createAndAdd(parent,
              flags.value_or(wxSizerFlags()));
        parent->SetSizerAndFit(sizer);
        return sizer;
    }
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
// Create the controls.
wxButton* button = new wxButton(this, wxID_ANY, "Click");
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
wxButton* done = new wxButton(this, wxID_EXIT, "Done");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
sizerTop->Add(button, wxSizerFlags().Border());
sizerTop->Add(text1, wxSizerFlags(1).Border());

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
sizerBottom->Add(text2, wxSizerFlags().Border());
sizerBottom->Add(done, wxSizerFlags().Border());

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
// Create the controls.
wxButton* button = new wxButton(this, wxID_ANY, "Click");
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
wxButton* done = new wxButton(this, wxID_EXIT, "Done");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
sizerTop->Add(button, wxSizerFlags().Border());
sizerTop->Add(text1, wxSizerFlags(1).Border());

sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
sizerBottom->Add(text2, wxSizerFlags().Border());
sizerBottom->Add(done, wxSizerFlags().Border());

sizer->Add(sizerBottom, wxSizerFlags().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

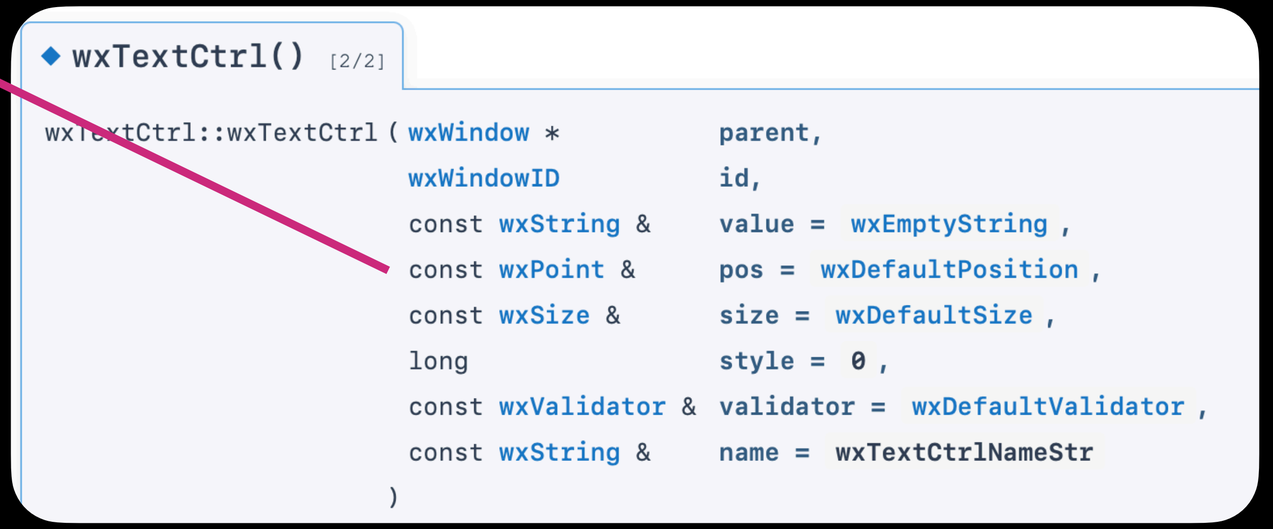
```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};
```

wxTextCtrl() [2/2]

```
wxTextCtrl::wxTextCtrl( wxWindow *           parent,
                        wxWindowID           id,
                        const wxString &     value = wxEmptyString,
                        const wxPoint &      pos = wxDefaultPosition,
                        const wxSize &       size = wxDefaultSize,
                        long                 style = 0,
                        const wxValidator &  validator = wxDefaultValidator,
                        const wxString &     name = wxTextCtrlNameStr
                      )
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_), flags_.value_or(suppliedFlags));
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    std::optional<wxSizerFlags> flags_;
};
```



```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> void
    {
        flags_ = flags;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog", wxSizerFlags(1).Border()
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> void
    {
        flags_ = flags;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Expand().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        wxSizerFlags().Border(),
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str,
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxS
    {
        sizer->Add(new W(parent, id_, str_,
    }

    auto withFlags(wxSizerFlags flags) -> v
    {
        flags_ = flags;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```
[main] Building folder: /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/build all
[build] Starting build
[proc] Executing command: /opt/homebrew/bin/cmake --build /Users/richardpowell/
Development/github.com/rmpowell77/wxHelloWorld.git/build --config Debug --target all --
[build] [1/2  50% :: 0.955] Building CXX object CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o
[build] FAILED: CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o
[build] /usr/bin/clang++ -DWXUSINGDLL -D_FILE_OFFSET_BITS=64 -D__WXMAC__
-D__WXOSX_COCOA__ -D__WXOSX__ -isystem /opt/homebrew/lib/wx/include/osx_cocoa-
unicode-3.2 -isystem /opt/homebrew/include/wx-3.2 -g -std=c++20 -arch arm64 -isysroot /
Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
MacOSX15.2.sdk -Wall -Wextra -Werror -MD -MT CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o -MF CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o.d -o CMakeFiles/
wxHelloWorld.dir/HelloWorld.cpp.o -c /Users/richardpowell/Development/github.com/
rmpowell77/wxHelloWorld.git/HelloWorld.cpp
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:208:9: error: no viable constructor or deduction guide for deduction of
template arguments of 'HSizer'
[build]   208 |         HSizer {
[build]       |         ^
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:120:5: note: candidate function template not viable: cannot convert
argument of incomplete type 'void' to 'wxSizerFlags' for 1st argument
[build]   120 |     HSizer(wxSizerFlags flags, W... widgets)
[build]       |     ^      ~~~~~~~~~~~~~~~~~~
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:116:5: note: candidate template ignored: substitution failure [with W =
<void, Button>]: argument may not have 'void' type
[build]   116 |     HSizer(W... widgets)
[build]       |     ^           ~~~~~~~
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:114:8: note: candidate function template not viable: requires 1 argument,
but 2 were provided
[build]   114 | struct HSizer : details::Sizer<W...> {
[build]       |        ^~~~~
[build] 1 error generated.
[build] ninja: build stopped: subcommand failed.
[proc] The command: /opt/homebrew/bin/cmake --build /Users/richardpowell/Development/
github.com/rmpowell77/wxHelloWorld.git/build --config Debug --target all -- exited with
code: 1
[driver] Build completed: 00:00:00.991
[build] Build finished with exit code 1
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> void
    {
        flags_ = flags;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> Widget<W>&
    {
        flags_ = flags;
        return *this;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

Fluent Builder Pattern

aka Method chaining

Fluent Builder Pattern

aka Method chaining

```
auto withSize(wxSize size_) -> Widget<W>&
{
    size = size_;
    return *this;
}
```

Fluent Builder Pattern

aka Method chaining

```
auto withSize(wxSize size_) -> Widget<W>&
{
    size = size_;
    return *this;
}
```

```
Button { "Exit" }
    .withColor(RED)
    .withWidth(16)
    .withHelp(),
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str, std::optional<wxSizerFlags> flags = {})
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> Widget<W>&
    {
        flags_ = flags;
        return *this;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

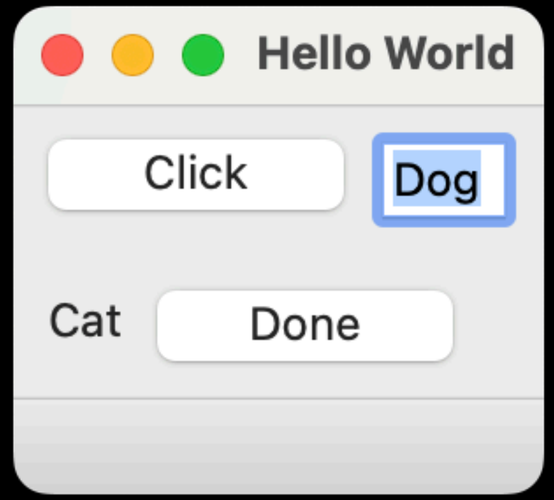
```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> Widget<W>&
    {
        flags_ = flags;
        return *this;
    }

private:
    wxWindowID id_ { wxID_ANY };
    std::string str_;
    wxPoint pos_ { wxDefaultPosition };
    wxSize size_ { wxDefaultSize };
    long style_ { 0 };
    std::optional<wxSizerFlags> flags_;
};
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
        , flags_(flags)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> Widget<W>&
    {
        flags_ = flags;
        return *this;
    }

    auto withFlags(wxSizerFlags flags) -> Widget&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Widget&
    {
        pos_ = pos;
        return *this;
    }
```

```cpp
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```



```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create the controls.
    wxButton* button = new wxButton(this, wxID_ANY, "Click");
    wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
    wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");
    wxButton* done = new wxButton(this, wxID_EXIT, "Done");
    wxSlider* slider = new wxSlider(this, wxID_ANY, 3, 1, 10);

    // Layout the controls.
    wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

    wxBoxSizer* sizerTop = new wxBoxSizer(wxHORIZONTAL);
    sizerTop->Add(button, wxSizerFlags().Border());
    sizerTop->Add(text1, wxSizerFlags(1).Border());

    sizer->Add(sizerTop, wxSizerFlags().Border().Expand());

    sizer->Add(slider, wxSizerFlags().Border().Expand());

    wxBoxSizer* sizerBottom = new wxBoxSizer(wxHORIZONTAL);
    sizerBottom->Add(text2, wxSizerFlags().Border());
    sizerBottom->Add(done, wxSizerFlags().Border());

    sizer->Add(sizerBottom, wxSizerFlags().Border());

    SetSizerAndFit(sizer);
}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

using TextCtrl = Widget<wxTextCtrl>;
using Button = Widget<wxButton>;
using Text = Widget<wxStaticText>;

}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider {},
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

using TextCtrl = Widget<wxTextCtrl>;
using Button = Widget<wxButton>;
using Text = Widget<wxStaticText>;
using Slider = Widget<wxSlider>;

}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider {},
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

using TextCtrl = Widget<wxTextCtrl>;
using Button = Widget<wxButton>;
using Text = Widget<wxStaticText>;
using Slider = Widget<wxSlider>;

}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).
    Slider {},
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
[main] Building folder: /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/build all
[build] Starting build
[proc] Executing command: /opt/homebrew/bin/cmake --build /Users/richardpowell/
Development/github.com/rmpowell77/wxHelloWorld.git/build --config Debug --target all --
[build] [1/2  50% :: 1.001] Building CXX object CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o
[build] FAILED: CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o
[build] /usr/bin/clang++ -DWXUSINGDLL -D_FILE_OFFSET_BITS=64 -D__WXMAC__
-D__WXOSX_COCOA__ -D__WXOSX__ -isystem /opt/homebrew/lib/wx/include/osx_cocoa-
unicode-3.2 -isystem /opt/homebrew/include/wx-3.2 -g -std=c++20 -arch arm64 -isysroot /
Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
MacOSX15.2.sdk -Wall -Wextra -Werror -MD -MT CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o -MF CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o.d -o CMakeFiles/
wxHelloWorld.dir/HelloWorld.cpp.o -c /Users/richardpowell/Development/github.com/
rmpowell77/wxHelloWorld.git/HelloWorld.cpp
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:50:24: error: no matching constructor for initialization of 'wxSlider'
[build]    50 |         sizer->Add(new W(parent, id, str, wxDefaultPosition, size),
flags ? *flags : parentFlags);
[build]       |                        ^ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:13:12: note: in instantiation of member function
'DeclarativeUI::details::Widget<wxSlider>::createAndAdd' requested here
[build]    13 |     widget.createAndAdd(window, sizer, wxSizerFlags {});
[build]       |            ^
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:13:5: note: in instantiation of requirement here
[build]    13 |     widget.createAndAdd(window, sizer, wxSizerFlags {});
[build]       |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:12:28: note: while substituting template arguments into constraint
expression here
[build]    12 | concept CreateAndAddable = requires(T widget, wxWindow* window, wxSizer*
sizer) {
[build]       |
^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build]    13 |     widget.createAndAdd(window, sizer, wxSizerFlags {});
[build]       |     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build]    14 | };
[build]       | ~
[build] /Users/richardpowell/Development/github.com/rmpowell77/wxHelloWorld.git/
HelloWorld.cpp:138:11: note: while checking the satisfaction of concept
'CreateAndAddable<DeclarativeUI::details::Widget<wxSlider>>' requested here
```

◆ wxTextCtrl() [2/2]

```
wxTextCtrl::wxTextCtrl ( wxWindow *           parent,
                         wxWindowID           id,
                         const wxString &     value =  wxEmptyString ,
                         const wxPoint &      pos =  wxDefaultPosition ,
                         const wxSize &       size =  wxDefaultSize ,
                         long                 style =  0 ,
                         const wxValidator &  validator =  wxDefaultValidator ,
                         const wxString &     name =  wxTextCtrlNameStr
                       )
```

◆ wxStaticText() [2/2]

```
wxStaticText::wxStaticText ( wxWindow *        parent,
                             wxWindowID        id,
                             const wxString &  label,
                             const wxPoint &   pos =  wxDefaultPosition ,
                             const wxSize &    size =  wxDefaultSize ,
                             long              style =  0 ,
                             const wxString &  name =  wxStaticTextNameStr
                           )
```

◆ wxButton() [2/2]

```
wxButton::wxButton ( wxWindow *           parent,
                     wxWindowID           id,
                     const wxString &     label =  wxEmptyString ,
                     const wxPoint &      pos =  wxDefaultPosition ,
                     const wxSize &       size =  wxDefaultSize ,
                     long                 style =  0 ,
```

wxTextCtrl() [2/2]

```
wxTextCtrl::wxTextCtrl ( wxWindow *           parent,
                         wxWindowID           id,
                         const wxString &     value = wxEmptyString ,
                         const wxPoint &      pos = wxDefaultPosition ,
                         const wxSize &       size = wxDefaultSize ,
                         long                 style = 0 ,
                         const wxValidator &  validator = wxDefaultValidator ,
                         const wxString &     name = wxTextCtrlNameStr
                       )
```

wxStaticText() [2/2]

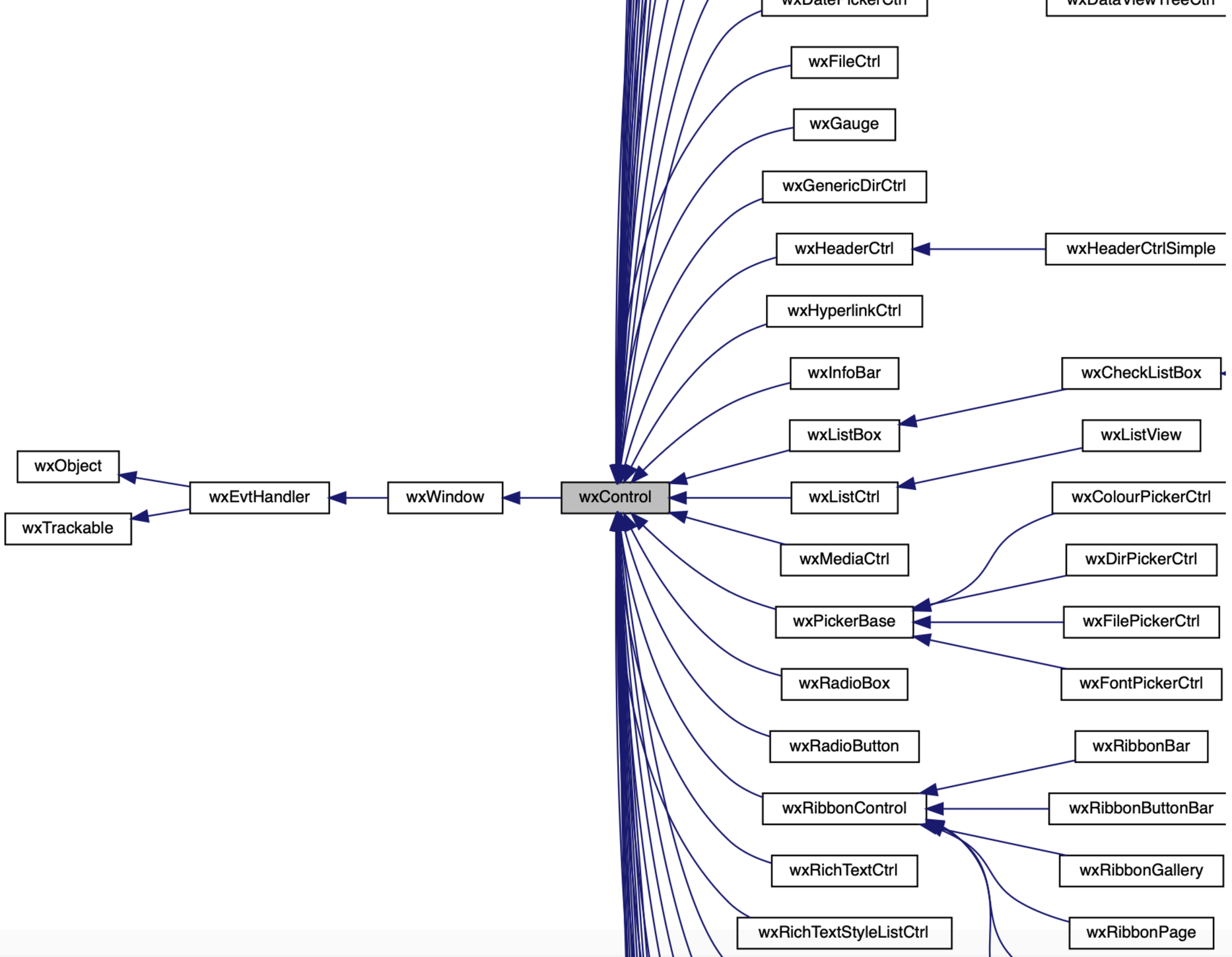
```
wxStaticText::wxStaticText ( wxWindow *        parent,
                             wxWindowID        id,
                             const wxString &  label,
                             const wxPoint &   pos = wxDefaultPosition ,
                             const wxSize &    size = wxDefaultSize ,
                             long              style = 0 ,
                             const wxString &  name = wxStaticTextNameStr
                           )
```

wxButton() [2/2]

```
wxButton::wxButton ( wxWindow *           parent,
                     wxWindowID           id,
                     const wxString &     label = wxEmptyString ,
                     const wxPoint &      pos = wxDefaultPosition ,
                     const wxSize &       size = wxDefaultSize ,
                     long                 style = 0 ,
```

wxSlider() [2/2]

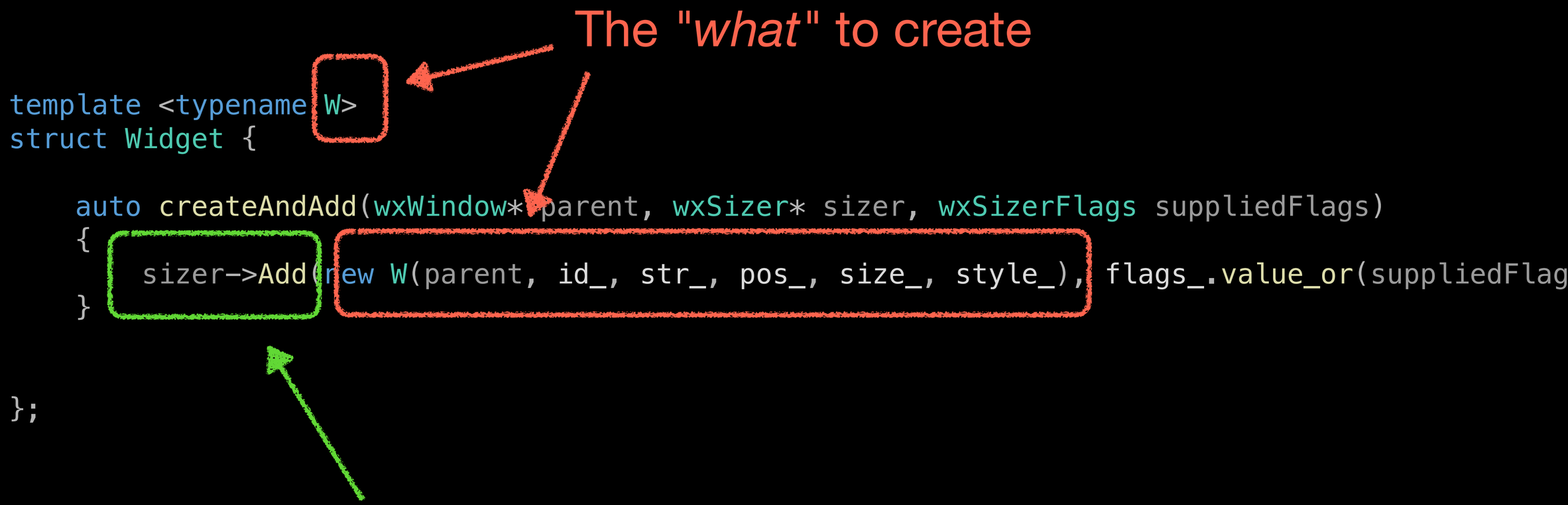
```
wxSlider::wxSlider ( wxWindow *           parent,
                     wxWindowID           id,
                     int                  value,
                     int                  minValue,
                     int                  maxValue,
                     const wxPoint &      pos = wxDefaultPosition ,
                     const wxSize &       size = wxDefaultSize ,
                     long                 style = wxSL_HORIZONTAL ,
                     const wxValidator &  validator = wxDefaultValidator ,
                     const wxString &     name = wxSliderNameStr
                   )
```


wxObject
wxTrackable
wxEvtHandler
wxWindow
wxControl
wxFileCtrl
wxGauge
wxGenericDirCtrl
wxHeaderCtrl
wxHeaderCtrlSimple
wxHyperlinkCtrl
wxInfoBar
wxCheckListBox
wxListBox
wxListView
wxListCtrl
wxColourPickerCtrl
wxMediaCtrl
wxDirPickerCtrl
wxPickerBase
wxFilePickerCtrl
wxRadioBox
wxFontPickerCtrl
wxRadioButton
wxRibbonBar
wxRibbonControl
wxRibbonButtonBar
wxRichTextCtrl
wxRibbonGallery
wxRichTextStyleListCtrl
wxRibbonPage

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlag
    }


};
```

The "*what*" to create

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlag
    }


};
```

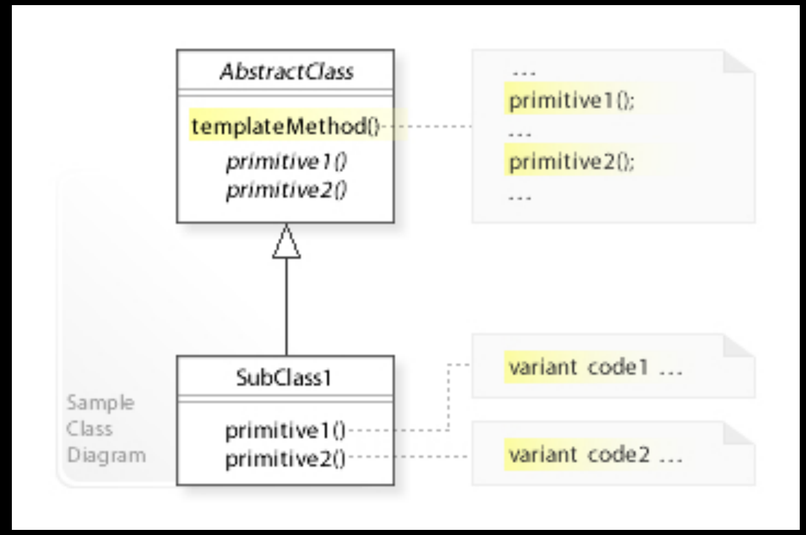

The "*what*" to create
template <typename W>
struct Widget {
auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
{
sizer->Add(new W(parent, id_, str_, pos_, size_, style_), flags_.value_or(suppliedFlag
}
};
The "*how*" to add

Template method pattern

The ***template method*** is a method in a superclass, and defines the skeleton of an operation in terms of a number of high-level steps.

These steps are implemented by additional *helper methods*.

Subclasses customize the operation by overriding the *helper methods*.



https://en.wikipedia.org/wiki/Template_method_pattern

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            new W(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:

};
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            new W(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
template <typename W>
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Text {

};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Text : Widget {
    using super = Widget;
    explicit Text(wxWindowID id = wxID_ANY, std::string str
        : super(id, std::move(str))
    {
    }

    explicit Text(std::string str)
        : Text(wxID_ANY, std::move(str))
    {
    }

};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Text : Widget {
    using super = Widget;
    explicit Text(wxWindowID id = wxID_ANY, std::string str
        : super(id, std::move(str))
    {
    }

    explicit Text(std::string str)
        : Text(wxID_ANY, std::move(str))
    {
    }

private:
    auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos, wxSize size,
        long style) -> wxWindow*
    {
        return new wxStaticText(parent, id, str, pos, size,
    }
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

The "*how*" to add

```
struct Text : Widget {
    using super = Widget;
    explicit Text(wxWindowID id = wxID_ANY, std::string str
        : super(id, std::move(str))
    {
    }

    explicit Text(std::string str)
        : Text(wxID_ANY, std::move(str))
    {
    }

private:
    auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos, wxSize size,
        long style) -> wxWindow*
    {
        return new wxStaticText(parent, id, str, pos, size,
    }
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

The "*how*" to add

The "*what*" to create

```
struct Text : Widget {
    using super = Widget;
    explicit Text(wxWindowID id = wxID_ANY, std::string str
        : super(id, std::move(str))
    {
    }

    explicit Text(std::string str)
        : Text(wxID_ANY, std::move(str))
    {
    }

private:
    auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos, wxSize size,
        long style) -> wxWindow*
    {
        return new wxStaticText(parent, id, str, pos, size,
    }
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_
            flags_.value_or(suppliedFlags)
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
[main] Building folder: /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/build
[build] Starting build
[proc] Executing command: /opt/homebrew/bin/cmake --build /Users/richardpowell/
Development/github.com/rmpowell77/wxHelloWorld.git/build --config Debug --target all --
[build] [1/2  50% :: 0.927] Building CXX object CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o
[build] FAILED: CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o
[build] /usr/bin/clang++ -DWXUSINGDLL -D_FILE_OFFSET_BITS=64 -D__WXMAC__
-D__WXOSX_COCOA__ -D__WXOSX__ -isystem /opt/homebrew/lib/wx/include/osx_cocoa-
unicode-3.2 -isystem /opt/homebrew/include/wx-3.2 -g -std=c++20 -arch arm64 -isysroot /
Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
MacOSX15.2.sdk -Wall -Wextra -Werror -MD -MT CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o -MF CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o.d -o CMakeFiles/
wxHelloWorld.dir/HelloWorld.cpp.o -c /Users/richardpowell/Development/github.com/
rmpowell77/wxHelloWorld.git/HelloWorld.cpp
[build] In file included from /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/HelloWorld.cpp:5:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/wx.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/object.h:19:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/memory.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/string.h:37:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/strvararg.h:19:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/unichar.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/stringimpl.h:66:
[build] In file included from /Applications/Xcode.app/Contents/Developer/Platforms/
MacOSX.platform/Developer/SDKs/MacOSX15.2.sdk/usr/include/c++/v1/string:594:
[build] In file included from /Applications/Xcode.app/Contents/Developer/Platforms/
MacOSX.platform/Developer/SDKs/MacOSX15.2.sdk/usr/include/c++/v1/__memory_resource/
polymorphic_allocator.h:20:
[build] /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX15.2.sdk/usr/include/c++/v1/tuple:306:7: error: field type
'DeclarativeUI::details::Widget' is an abstract class
[build]   306 |   _Hp __value_;
[build]       |       ^
[build] /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX15.2.sdk/usr/include/c++/v1/tuple:464:14: note: in instantiation of template
class 'std::__tuple_leaf<0, DeclarativeUI::details::Widget>' requested here
[build]   464 |     : public __tuple_leaf<_Indx, _Tp>... {
[build]       |              ^
[build] /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX15.2.sdk/usr/include/c++/v1/tuple:536:10: note: in instantiation of template
class 'std::__tuple_impl<std::__tuple_indices<0, 1>, DeclarativeUI::details::Widget,
DeclarativeUI::Button>' requested here
```

```cpp
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_
            flags_.value_or(suppliedFlags)
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
[main] Building folder: /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/build
[build] Starting build
[proc] Executing command: /opt/homebrew/bin/cmake --build /Users/richardpowell/
Development/github.com/rmpowell77/wxHelloWorld.git/build --config Debug --target all --
[build] [1/2  50% :: 0.927] Building CXX object CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o
[build] FAILED: CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o
[build] /usr/bin/clang++ -DWXUSINGDLL -D_FILE_OFFSET_BITS=64 -D__WXMAC__
-D__WXOSX_COCOA__ -D__WXOSX__ -isystem /opt/homebrew/lib/wx/include/osx_cocoa-
unicode-3.2 -isystem /opt/homebrew/include/wx-3.2 -g -std=c++20 -arch arm64 -isysroot /
Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
MacOSX15.2.sdk -Wall -Wextra -Werror -MD -MT CMakeFiles/wxHelloWorld.dir/
HelloWorld.cpp.o -MF CMakeFiles/wxHelloWorld.dir/HelloWorld.cpp.o.d -o CMakeFiles/
wxHelloWorld.dir/HelloWorld.cpp.o -c /Users/richardpowell/Development/github.com/
rmpowell77/wxHelloWorld.git/HelloWorld.cpp
[build] In file included from /Users/richardpowell/Development/github.com/rmpowell77/
wxHelloWorld.git/HelloWorld.cpp:5:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/wx.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/object.h:19:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/memory.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/string.h:37:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/strvararg.h:19:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/unichar.h:15:
[build] In file included from /opt/homebrew/include/wx-3.2/wx/stringimpl.h:66:
[build] In file included from /Applications/Xcode.app/Contents/Developer/Platforms/
MacOSX.platform/Developer/SDKs/MacOSX15.2.sdk/usr/include/c++/v1/string:594:
[build] In file included from /Applications/Xcode.app/Contents/Developer/Platforms/
MacOSX.platform/Developer/SDKs/MacOSX15.2.sdk/usr/include/c++/v1/__memory_resource/
polymorphic_allocator.h:20:
[build] /Applications/Xcode.app/Contents/Developer/Platforms/
MacOSX.platform/Developer/SDKs/MacOSX15.2.sdk/usr/include/c++/v1/
tuple:306:7: error: field type 'DeclarativeUI::details::Widget'
is an abstract class
[build]   306 |   _Hp __value_;
[build]       |       ^
[build] /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX15.2.sdk/usr/include/c++/v1/tuple:464:14: note: in instantiation of template
class 'std::__tuple_leaf<0, DeclarativeUI::details::Widget>' requested here
[build]   464 |     : public __tuple_leaf<_Indx, _Tp>... {
[build]       |              ^
[build] /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX15.2.sdk/usr/include/c++/v1/tuple:536:10: note: in instantiation of template
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Widget {
    Widget(wxWindowID id, std::string str)
        : id_(id)
        , str_(std::move(str))
    {
    }

    auto createAndAdd(
        wxWindow* parent,
        wxSizer* sizer,
        wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            create(parent, id_, str_, pos_, size_, style_),
            flags_.value_or(suppliedFlags));
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Widget&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Widget&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Widget&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Widget&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Widget&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Widget&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Widget&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Widget&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Widget&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Widget&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Widget&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Widget&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> TextCtrl&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> TextCtrl&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> TextCtrl&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct TextCtrl : Widget;
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Button&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Button&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Button&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Button : Widget;
```

```
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Button&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Button&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Button&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Button : Widget;
```

```
template <typename W>
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Button&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Button&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Button&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```
struct Button : Widget;
```

```cpp
template <typename W>
struct Widget {
    auto withFlags(wxSizerFlags flags) -> Button&
    {
        flags_ = flags;
        return *this;
    }

    auto withPos(wxPoint pos) -> Button&
    {
        pos_ = pos;
        return *this;
    }

    auto withSize(wxSize size) -> Button&
    {
        size_ = size;
        return *this;
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```cpp
struct Button : Widget<Button>;
```

```cpp
template <typename W>
struct Widget {
    auto withFlags(wxSizerFlags flags) -> W&
    {
        flags_ = flags;
        return static_cast<W&>(*this);
    }

    auto withPos(wxPoint pos) -> W&
    {
        pos_ = pos;
        return static_cast<W&>(*this);
    }

    auto withSize(wxSize size) -> W&
    {
        size_ = size;
        return static_cast<W&>(*this);
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```cpp
struct Button : Widget<Button>;
```

```cpp
template <typename W>
struct Widget {
    auto withFlags(wxSizerFlags flags) -> W&
    {
        flags_ = flags;
        return static_cast<W&>(*this);
    }

    auto withPos(wxPoint pos) -> W&
    {
        pos_ = pos;
        return static_cast<W&>(*this);
    }

    auto withSize(wxSize size) -> W&
    {
        size_ = size;
        return static_cast<W&>(*this);
    }

private:
    virtual auto create(
        wxWindow* parent,
        wxWindowID id,
        std::string const& str,
        wxPoint pos,
        wxSize size,
        long style) -> wxWindow* = 0;
};
```

```cpp
struct Button : Widget<Button>;
```

```cpp
template <typename W>
struct Widget {
private:
    wxWindowID id;
    wxPoint position = wxDefaultPosition;
    wxSize size = wxDefaultSize;
    std::string str;
    std::optional<wxSizerFlags> flags;
    std::optional<Handler> boundedHandler;
};
```

```cpp
struct Text : Widget<Text> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::str
    {
        return new wxStaticText(parent, id, str, pos, siz
    }

};
```

```cpp
struct TextCtrl : Widget<TextCtrl> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::str
    {
        return new wxTextCtrl(parent, id, str, pos, size,
    }

};
```

```cpp
struct Button : Widget<Button> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::st
    {
        return new wxButton(parent, id, str, pos, size,
    }

};
```

```cpp
template <typename W>
struct Widget {
private:
    wxWindowID id;
    wxPoint position = wxDefaultPosition;
    wxSize size = wxDefaultSize;
    std::optional<wxSizerFlags> flags;
    std::optional<Handler> boundedHandler;
};
```

```cpp
struct Text : details::Widget<Text> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::str
    {
        return new wxStaticText(parent, id, str, pos, siz
    }
    std::string str;
};
```

```cpp
struct TextCtrl : details::Widget<TextCtrl> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::str
    {
        return new wxTextCtrl(parent, id, str, pos, size,
    }
    std::string str;
};
```

```cpp
struct Button : details::Widget<Button> {
private:
    auto create(wxWindow* parent, wxWindowID id, std::st
    {
        return new wxButton(parent, id, str, pos, size,
    }
    std::string str;
};
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

struct TextCtrl : Widget<TextCtrl>;
struct Button : Widget<Button>;
struct Text : Widget<Text>;

}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider {},
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

struct TextCtrl : Widget<TextCtrl>;
struct Button : Widget<Button>;
struct Text : Widget<Text>;
struct Slider : Widget<Slider>;

}
```

```
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider {},
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

???

```
namespace DeclarativeUI {

struct TextCtrl : Widget<TextCtrl>;
struct Button : Widget<Button>;
struct Text : Widget<Text>;
struct Slider : Widget<Slider>;

}
```

wxSlider() [2/2]

```
wxSlider::wxSlider ( wxWindow *          parent,
                     wxWindowID          id,
                     int                 value,
                     int                 minValue,
                     int                 maxValue,
                     const wxPoint &     pos =  wxDefaultPosition ,
                     const wxSize &      size =  wxDefaultSize ,
                     long                style =  wxSL_HORIZONTAL ,
                     const wxValidator & validator =  wxDefaultValidator ,
                     const wxString &    name =  wxSliderNameStr
                   )
```

◆ wxSlider() [2/2]

```
wxSlider::wxSlider ( wxWindow *          parent,
                     wxWindowID          id,
                     int                 value,
                     int                 minValue,
                     int                 maxValue,
                     const wxPoint &     pos = wxDefaultPosition ,
                     const wxSize &      size = wxDefaultSize ,
                     long                style = wxSL_HORIZONTAL ,
                     const wxValidator & validator = wxDefaultValidator ,
                     const wxString &    name = wxSliderNameStr
                   )
```

◆ wxSlider() [2/2]

```
wxSlider::wxSlider ( wxWindow *           parent,
                     wxWindowID           id,
                     int                  value,
                     int                  minValue,
                     int                  maxValue,
                     const wxPoint &      pos = wxDefaultPosition ,
                     const wxSize &       size = wxDefaultSize ,
                     long                 style = wxSL_HORIZONTAL ,
                     const wxValidator &  validator = wxDefaultValidator ,
                     const wxString &     name = wxSliderNameStr
                   )
```

```
typedef int wxWindowID;
```

◆ wxSlider() [2/2]

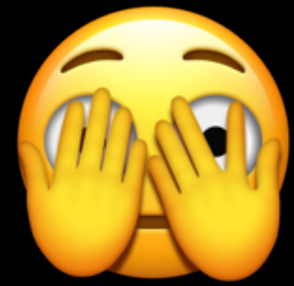
```
wxSlider::wxSlider ( wxWindow *          parent,
                     wxWindowID          id,
                     int                 value,
                     int                 minValue,
                     int                 maxValue,
                     const wxPoint &     pos = wxDefaultPosition ,
                     const wxSize &      size = wxDefaultSize ,
                     long                style = wxSL_HORIZONTAL ,
                     const wxValidator & validator = wxDefaultValidator ,
                     const wxString &    name = wxSliderNameStr
                   )
```

```
typedef int wxWindowID;
```

◆ wxSlider() [2/2]

```
wxSlider::wxSlider( wxWindow *          parent,
                    wxWindowID          id,
                    int                 value,
                    int                 minValue,
                    int                 maxValue,
                    const wxPoint &     pos = wxDefaultPosition,
                    const wxSize &      size = wxDefaultSize,
                    long                style = wxSL_HORIZONTAL,
                    const wxValidator & validator = wxDefaultValidator,
                    const wxString &    name = wxSliderNameStr
                  )
```

```
typedef int wxWindowID;
```



wxSlider() [2/2]

```
wxSlider::wxSlider ( wxWindow *          parent,
                     wxWindowID          id,
                     int                 value,
                     int                 minValue,
                     int                 maxValue,
                     const wxPoint &     pos =  wxDefaultPosition ,
                     const wxSize &      size =  wxDefaultSize ,
                     long                style =  wxSL_HORIZONTAL ,
                     const wxValidator & validator =  wxDefaultValidator ,
                     const wxString &    name =  wxSliderNameStr
                   )
```

wxSlider() [2/2]

```
wxSlider::wxSlider ( wxWindow *          parent,
                     wxWindowID          id,
                     int                 value,
                     int                 minValue,
                     int                 maxValue,
                     const wxPoint &     pos = wxDefaultPosition ,
                     const wxSize &      size = wxDefaultSize ,
                     long                style = wxSL_HORIZONTAL ,
                     const wxValidator & validator = wxDefaultValidator ,
                     const wxString &    name = wxSliderNameStr
                   )
```

```
Slider(std::pair<int, int> range);
Slider(std::pair<int, int> range, int value);
Slider(wxWindowID id, std::pair<int, int> range);
Slider(wxWindowID id, std::pair<int, int> range, int value);
```

```
struct Slider : Widget<Slider> {
    using super = Widget<Slider>;
    Slider(wxWindowID id, std::pair<int, int> range, std::optional<int> min = std::nullopt)
        : super(id)
        , range_(range)
        , min_(min.value_or(range.first))
    {
    }

    explicit Slider(std::pair<int, int> range, std::optional<int> min = std::nullopt)
        : Slider(wxID_ANY, range, min)
    {
    }

private:
    auto create(wxWindow* parent, wxWindowID id, wxPoint pos, wxSize size, long style) -> wxWindow*
    {
        return new wxSlider(parent, id, min_, range_.first, range_.second, pos, size, style);
    }
    std::pair<int, int> range_;
    int min_;
};
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" },
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider {},
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
namespace DeclarativeUI {

struct TextCtrl : Widget<TextCtrl>;
struct Button : Widget<Button>;
struct Text : Widget<Text>;
struct Slider : Widget<Slider>;

}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" },
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 },
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
namespace DeclarativeUI {

struct TextCtrl : Widget<TextCtrl>;
struct Button : Widget<Button>;
struct Text : Widget<Text>;
struct Slider : Widget<Slider>;

}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" },
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 },
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```cpp
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{

    SetSizerAndFit(sizer);

    wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
    wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");

    // Lotsa code

    // And connect the event handlers.
    btnLeft->Bind(wxEVT_BUTTON, [](wxEvent&) { wxLogMessage("Pressed Left"); });
    btnRight->Bind(wxEVT_BUTTON, [](wxEvent&) { wxLogMessage("Pressed Right"); });
}
```

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(create(parent, id_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

private:

};
```

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(create(parent, id_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    using Handler = std::function<void(wxCommandEvent&)>;

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

};
```

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(create(parent, id_, pos_, size_, style_), flags_.value_or(suppliedFlags));
    }

    using Handler = std::function<void(wxCommandEvent&)>;

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            bindHandlers(create(parent, id_, pos_, size_, style_)),
            flags_.value_or(suppliedFlags));
    }

    using Handler = std::function<void(wxCommandEvent&)>;

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            bindHandlers(create(parent, id_, pos_, size_, style_)),
            flags_.value_or(suppliedFlags));
    }

    using Handler = std::function<void(wxCommandEvent&)>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler handler) -> W&
    {
        boundedHandlers[event] = handler;
        return static_cast<W&>(*this);
    }

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```cpp
template <typename W>
struct Widget {

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSiz
    {
        sizer->Add(
            bindHandlers(create(parent, id_, pos_, size_, st
            flags_.value_or(suppliedFlags));
    }

    using Handler = std::function<void(wxCommandEvent&)>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler
    {
        boundedHandlers[event] = handler;
        return static_cast<W&>(*this);
    }

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> bounde

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```cpp
struct Button : Widget<Button> {

    auto bind(Handler handler) -> Button&
    {
        return super::bind(wxEVT_BUTTON, handler);
    }

};

struct Slider : Widget<Slider> {

    auto bind(Handler handler) -> Slider&
    {
        return super::bind(wxEVT_SLIDER, handler);
    }

};
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" },
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 },
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([](wxCommandEvent&) {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([](wxCommandEvent&) {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([](wxCommandEvent&) {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([](wxCommandEvent&) {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```cpp
template <typename W>
struct Widget {

    using Handler = std::function<void(wxCommandEvent&)>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler handler) -> W&
    {
        boundedHandlers[event] = handler;
        return static_cast<W&>(*this);
    }

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```cpp
template <typename W>
struct Widget {

    using Handler = std::variant<std::function<void(wxCommandEvent&)>, std::function<void()>>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler handler) -> W&
    {
        boundedHandlers[event] = handler;
        return static_cast<W&>(*this);
    }

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            widget->Bind(event, func);
        }
        return widget;
    }
};
```

```cpp
template <typename W>
struct Widget {

    using Handler = std::variant<std::function<void(wxCommandEvent&)>, std::function<void()>>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler handler) -> W&
    {
        boundedHandlers[event] = handler;
        return static_cast<W&>(*this);
    }

private:
    std::map<wxEventTypeTag<wxCommandEvent>, Handler> boundedHandlers;

    auto bindHandlers(wxWindow* widget) -> wxWindow*
    {
        for (auto&& [event, func] : boundedHandlers) {
            std::visit(
                overloaded {
                    [&widget, event](std::function<void(wxCommandEvent&)> func) {
                        widget->Bind(event, func);
                    },
                    [&widget, event](std::function<void()> func) {
                        widget->Bind(event, [func](wxCommandEvent&) {
                            func();
                        });
                    },
                },
                func);
        }
        return widget;
    }
};
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([](wxCommandEvent&) {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
    : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    CreateStatusBar(1);
    // Create and layout the controls.
    VSizer {
        wxSizerFlags().Border(),
        HSizer {
            wxSizerFlags().Border().Expand(),
            Button { wxID_ANY, "Click" }.bind([] {
                wxLogMessage("Button Clicked!");
            }),
            TextCtrl { wxID_ANY, "Dog" }
                .withFlags(wxSizerFlags(1).Border()) },
        Slider { { 1, 10 }, 3 }.bind([this](wxCommandEvent& event) {
            SetStatusText(std::to_string(event.GetInt()));
        }),
        HSizer {
            Text { wxID_ANY, "Cat" },
            Button { wxID_EXIT, "Done" } }
    }.fitTo(this);
}
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
namespace DeclarativeUI {

template <typename T>
concept CreateAndAddable = requires(T widget, wxWindow* window, wxSizer* sizer) {
    widget.createAndAdd(window, sizer, wxSizerFlags {});
};

template <typename W>
struct Widget {
    explicit Widget(wxWindowID id)
        : id_(id)
    {
    }

    auto createAndAdd(wxWindow* parent, wxSizer* sizer, wxSizerFlags suppliedFlags)
    {
        sizer->Add(
            bindHandlers(create(parent, id_, pos_, size_, style_)),
            flags_.value_or(suppliedFlags));
    }

    auto withFlags(wxSizerFlags flags) -> W&
    {
        flags_ = flags;
        return static_cast<W&>(*this);
    }

    auto withPos(wxPoint pos) -> W&
    {
        pos_ = pos;
        return static_cast<W&>(*this);
    }

    auto withSize(wxSize size) -> W&
    {
        size_ = size;
        return static_cast<W&>(*this);
    }

    auto withWidth(int width) -> W&
    {
        size_.SetWidth(width);
        return static_cast<W&>(*this);
    }

    auto withHeight(int height) -> W&
    {
        size_.SetHeight(height);
        return static_cast<W&>(*this);
    }

    auto withStyle(long style) -> W&
    {
        style_ = style;
        return static_cast<W&>(*this);
    }

    using Handler = std::function<void(wxCommandEvent&)>;

protected:
    auto bind(wxEventTypeTag<wxCommandEvent> event, Handler handler) -> W&
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
    HSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxHORIZONTAL, flags, widgets...)
    {
    }
};

template <CreateAndAddable... W>
struct VSizer : Sizer<W...> {
    VSizer(W... widgets)
        : Sizer<W...>(wxVERTICAL, widgets...)
    {
    }
    VSizer(wxSizerFlags flags, W... widgets)
        : Sizer<W...>(wxVERTICAL, flags, widgets...)
    {
    }
};

struct TextCtrl : Widget<TextCtrl> {
    using super = Widget<TextCtrl>;
    explicit TextCtrl(wxWindowID id = wxID_ANY, std::string str = std::string {})
        : super(id)
        , str_(std::move(str))
    {
    }

    explicit TextCtrl(std::string str)
        : TextCtrl(wxID_ANY, std::move(str))
    {
    }

private:
    auto create(wxWindow* parent, wxWindowID id, wxPoint pos, wxSize size, long style) -> wxWindow*
    {
        return new wxTextCtrl(parent, id, str_, pos, size, style);
    }
    std::string str_;
};

struct Button : Widget<Button> {
    using super = Widget<Button>;
    explicit Button(wxWindowID id = wxID_ANY, std::string str = std::string {})
        : super(id)
        , str_(std::move(str))
    {
    }

    explicit Button(std::string str)
        : Button(wxID_ANY, std::move(str))
    {
    }

    auto bind(Handler handler) -> Button&
    {
        return super::bind(wxEVT_BUTTON, handler);
    }

private:
    auto create(wxWindow* parent, wxWindowID id, wxPoint pos, wxSize size, long style) -> wxWindow*
    {
        return new wxButton(parent, id, str_, pos, size, style);
    }
    std::string str_;
};

struct Text : Widget<Text> {
    using super = Widget<Text>;
    explicit Text(wxWindowID id = wxID_ANY, std::string str = std::string {})
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
// Create the controls.
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerText = new wxBoxSizer(wxHORIZONTAL);
sizerText->Add(text1, wxSizerFlags(1).Expand().Border());
sizerText->Add(btnRight, wxSizerFlags().Expand().Border());

sizer->Add(sizerText, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Expand().Border());
sizerBtns->Add(text2, wxSizerFlags().Expand().Border());

sizer->Add(sizerBtns, wxSizerFlags().Expand().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border())
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

Combining Controllers and Layout makes it easier to see intention.

```
// Create the controls.
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerText = new wxBoxSizer(wxHORIZONTAL);
sizerText->Add(text1, wxSizerFlags(1).Expand().Border());
sizerText->Add(btnRight, wxSizerFlags().Expand().Border());

sizer->Add(sizerText, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Expand().Border());
sizerBtns->Add(text2, wxSizerFlags().Expand().Border());

sizer->Add(sizerBtns, wxSizerFlags().Expand().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border())
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

Impossible to forget to add a controller to the layout.

```
// Create the controls.
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerText = new wxBoxSizer(wxHORIZONTAL);
sizerText->Add(text1, wxSizerFlags(1).Expand().Border());
sizerText->Add(btnRight, wxSizerFlags().Expand().Border());

sizer->Add(sizerText, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Expand().Border());
sizerBtns->Add(text2, wxSizerFlags().Expand().Border());

sizer->Add(sizerBtns, wxSizerFlags().Expand().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()),
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

Adding things to an incorrect Sizer greatly reduced.

```
// Create the controls.
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerText = new wxBoxSizer(wxHORIZONTAL);
sizerText->Add(text1, wxSizerFlags(1).Expand().Border());
sizerText->Add(btnRight, wxSizerFlags().Expand().Border());

sizer->Add(sizerText, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Expand().Border());
sizerBtns->Add(text2, wxSizerFlags().Expand().Border());

sizer->Add(sizerBtns, wxSizerFlags().Expand().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()),
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

Very "DRY"

```
// Create the controls.
wxTextCtrl* text1 = new wxTextCtrl(this, wxID_ANY, "Dog");
wxButton* btnRight = new wxButton(this, wxID_ANY, "Right");
wxButton* btnLeft = new wxButton(this, wxID_ANY, "Left");
wxStaticText* text2 = new wxStaticText(this, wxID_ANY, "Cat");

// Layout the controls.
wxBoxSizer* sizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer* sizerText = new wxBoxSizer(wxHORIZONTAL);
sizerText->Add(text1, wxSizerFlags(1).Expand().Border());
sizerText->Add(btnRight, wxSizerFlags().Expand().Border());

sizer->Add(sizerText, wxSizerFlags().Expand().Border());

wxBoxSizer* sizerBtns = new wxBoxSizer(wxHORIZONTAL);
sizerBtns->Add(btnLeft, wxSizerFlags().Expand().Border());
sizerBtns->Add(text2, wxSizerFlags().Expand().Border());

sizer->Add(sizerBtns, wxSizerFlags().Expand().Border());

SetSizerAndFit(sizer);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()),
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

- Look for the data structures hidden in your code.

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()),
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

- Look for the data structures hidden in your code.
- Reduce the number of statements. Favor expressions.

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

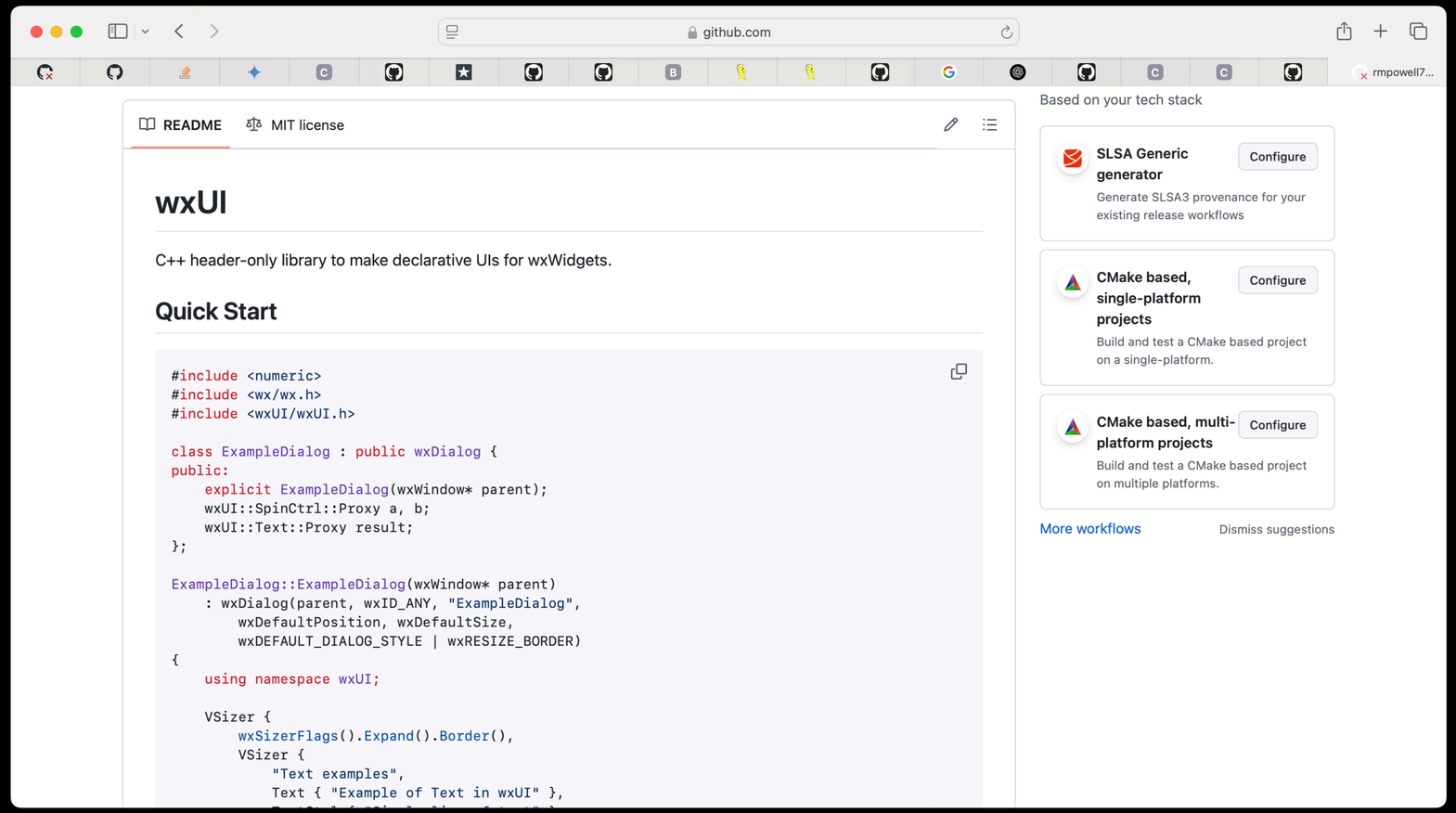
- Look for the data structures hidden in your code.
- Reduce the number of statements. Favor expressions.
- Builder Patterns allows for Composability

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()) },
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

- Look for the data structures hidden in your code.
- Reduce the number of statements. Favor expressions.
- Builder Patterns allows for Composability
- Template Pattern allows you to add flexibility and customization

```
using namespace DeclarativeUI;
// Create and layout the controls.
VSizer {
    wxSizerFlags().Border(),
    HSizer {
        wxSizerFlags().Border().Expand(),
        Button { wxID_ANY, "Click" },
        TextCtrl { wxID_ANY, "Dog" }
            .withFlags(wxSizerFlags(1).Border()),
    Slider { { 1, 10 }, 3 },
    HSizer {
        Text { wxID_ANY, "Cat" },
        Button { wxID_EXIT, "Done" } }
}.fitTo(this);
```

wxUI


github.com
README
MIT license
wxUI
C++ header-only library to make declarative UIs for wxWidgets.
Quick Start
#include <numeric>
#include <wx/wx.h>
#include <wxUI/wxUI.h>
class ExampleDialog : public wxDialog {
public:
explicit ExampleDialog(wxWindow* parent);
wxUI::SpinCtrl::Proxy a, b;
wxUI::Text::Proxy result;
};
ExampleDialog::ExampleDialog(wxWindow* parent)
: wxDialog(parent, wxID_ANY, "ExampleDialog",
wxDefaultPosition, wxDefaultSize,
wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
using namespace wxUI;
VSizer {
wxSizerFlags().Expand().Border(),
VSizer {
"Text examples",
Text { "Example of Text in wxUI" },
Based on your tech stack
SLSA Generic generator
Configure
Generate SLSA3 provenance for your existing release workflows
CMake based, single-platform projects
Configure
Build and test a CMake based project on a single-platform.
CMake based, multi-platform projects
Configure
Build and test a CMake based project on multiple platforms.
More workflows
Dismiss suggestions

Thank You