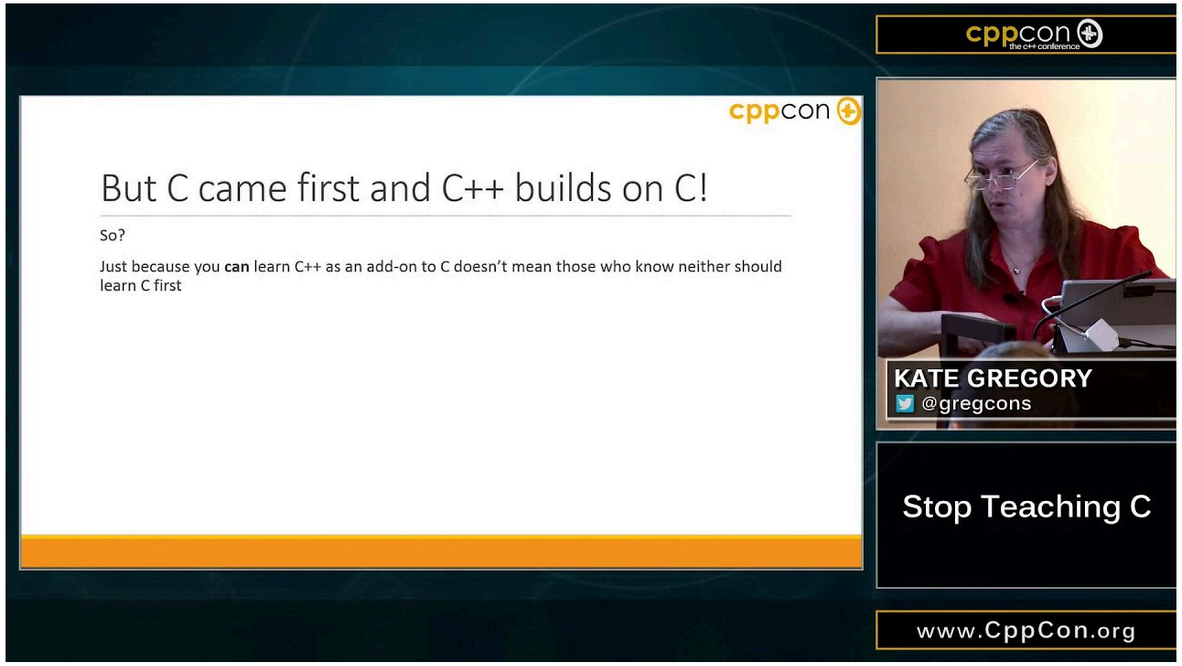
Cpp North 2025

Debugger Visualizers to Make Your Code Accessible

Braden Ganetsky



"Stop Teaching C"

https://www.youtube.com/watch?v=YnWhqhNdYyk

Scenario: Your program hits an error and exits

- No knowledge of where it happened
- Logs are too sparse to determine what happened
- What do you do?

"`printf` debugging"

- Modify the source code with a print statement
- Recompile the program
- Rerun the program, looking for the message
- Realize the print statement is wrong or insufficient
- Loop the above steps ad infinitum
- All the while, recompiling on every loop
- Big waste of time

Possible alternatives

- Don't write bugs?
- Use sanitizers to reduce bugs?
- Others...?
- Use a debugger to explore and walk through the code!

Using a debugger: The pros

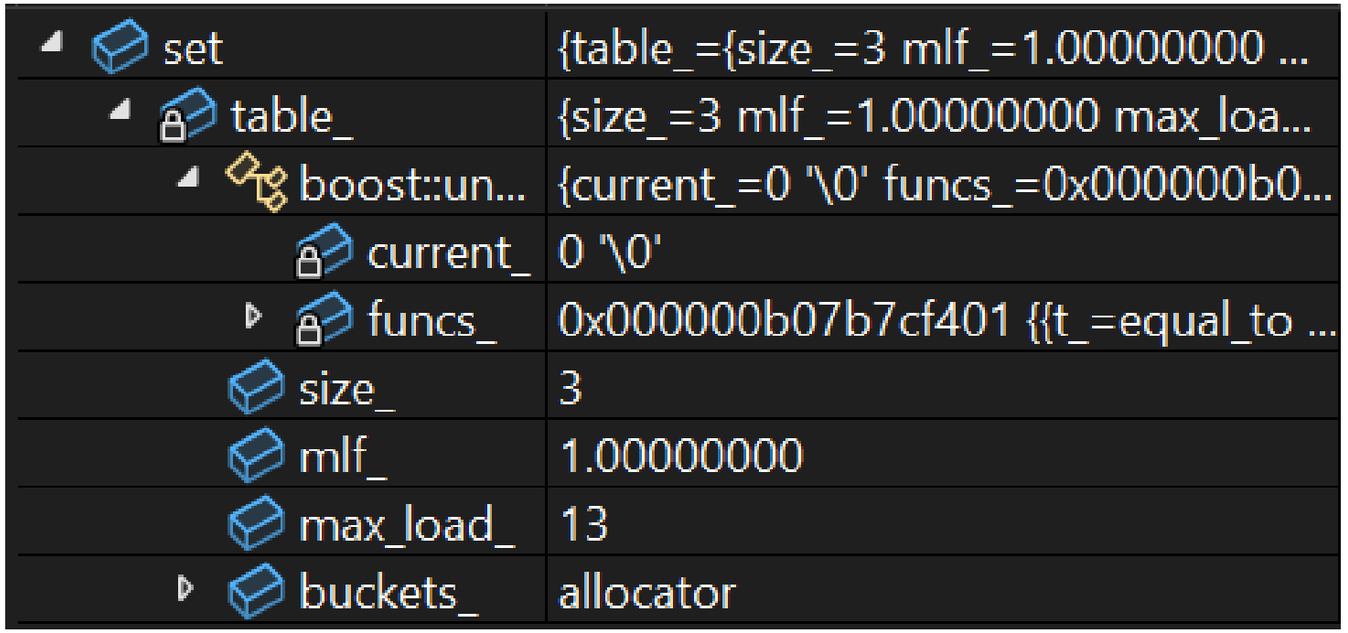
- No recompiling
- Iterating is much faster
- As granular as you want, e.g. stepping into function calls
- Special features, e.g. time travel debugging

Using a debugger: The cons

- More painful to set up
- Learning curve
- Less accessible to get started
- Incomprehensible data displayed

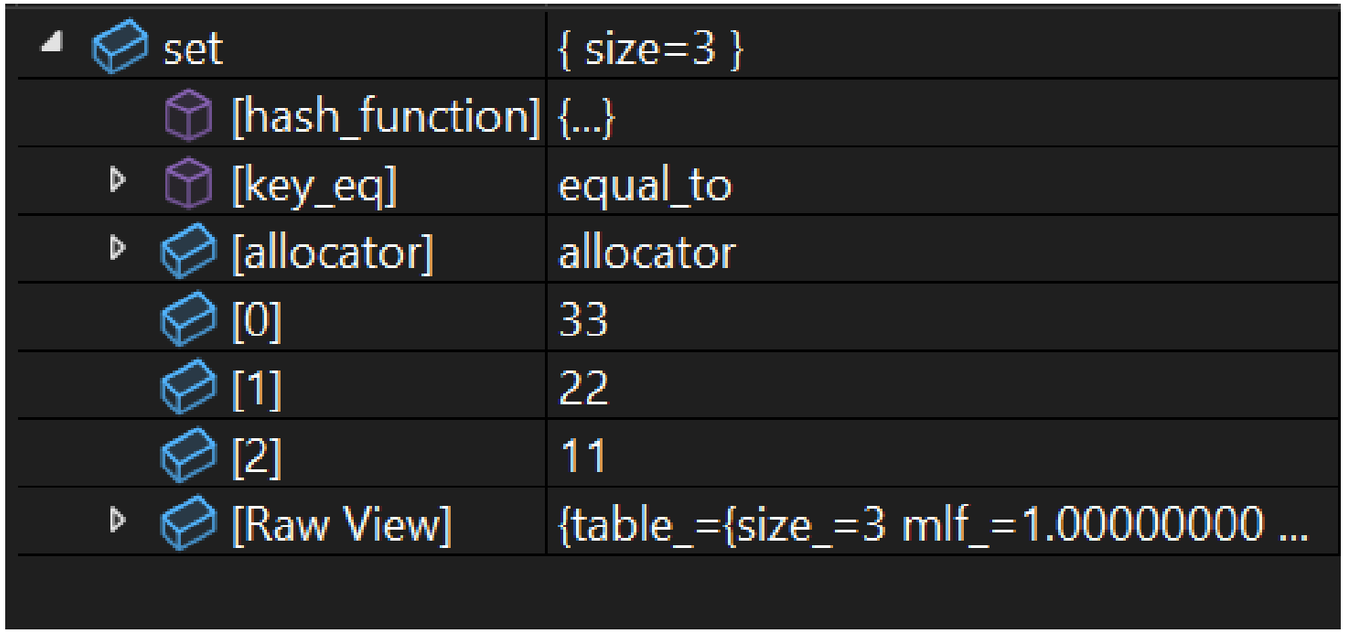
Incomprehensible data

```cpp
#include <boost/unordered_set.hpp>
int main() {
    boost::unordered_set<int> set{ 11, 22, 33 };
}
```



Incomprehensible data

```cpp
#include <boost/unordered_set.hpp>
int main() {
    boost::unordered_set<int> set{ 11, 22, 33 };
}
```



Incomprehensible data

```
#include <boost/unordered_set.hpp>
int main() {
    boost::unordered_set<int> set{ 11, 22, 33 };
}
```

```
(gdb) print set
$1 = {
  table_ = {
    <boost::unordered::detail::functions<boost::hash<int>, std::equal_to<int> >> = {
      static nothrow_move_assignable = true,
      static nothrow_move_constructible = true,
      static nothrow_swappable = true,
      current_ = 0 '\000',
      funcs_ = {{
          t_ = {
            <boost::unordered::detail::compressed_base<boost::hash<int>, 1>> = {
              <boost::empty_::empty_value<boost::hash<int>, 0u, true>> = {
                <boost::hash<int>> = {<No data fields>}, <No data fields>}, <No data fields>},
            <boost::unordered::detail::compressed_base<std::equal_to<int>, 2>> = {
              <boost::empty_::empty_value<std::equal_to<int>, 0u, true>> = {
                <std::equal_to<int>> = {
                  <std::binary_function<int, int, bool>> = {<No data fields>}, <No data fields>}, <No data fields>}, <No data fields>}
        }, {
          t_ = {
            <boost::unordered::detail::compressed_base<boost::hash<int>, 1>> = {
              <boost::empty_::empty_value<boost::hash<int>, 0u, true>> = {
                <boost::hash<int>> = {<No data fields>}, <No data fields>}, <No data fields>},
            <boost::unordered::detail::compressed_base<std::equal_to<int>, 2>> = {
              <boost::empty_::empty_value<std::equal_to<int>, 0u, true>> = {
```

```
                <std::equal_to<int>> = {
                  <std::binary_function<int, int, bool>> = {<No data fields>}, <No data fields>}, <No data fields>}, <No data fields>}
        }}
    },
    members of boost::unordered::detail::table<boost::unordered::detail::set<std::allocator<int>, int, boost::hash<int>, std::equal_to
    size_ = 3,
    mlf_ = 1,
    max_load_ = 13,
    buckets_ = {
      <boost::empty_::empty_value<std::allocator<boost::unordered::detail::node<int, void*> >, 0u, true>> = {
        <std::allocator<boost::unordered::detail::node<int, void*> >> = {
          <__gnu_cxx::new_allocator<boost::unordered::detail::node<int, void*> >> = {<No data fields>}, <No data fields>}, <No data fie
      members of boost::unordered::detail::grouped_bucket_array<boost::unordered::detail::bucket<boost::unordered::detail::node<int, vo
      size_index_ = 0,
      size_ = 13,
      buckets = 0x55555556feb0,
      groups = 0x55555556ff30
    }
  }
}
```

Incomprehensible data

```cpp
#include <boost/unordered_set.hpp>
int main() {
    boost::unordered_set<int> set{ 11, 22, 33 };
}
```

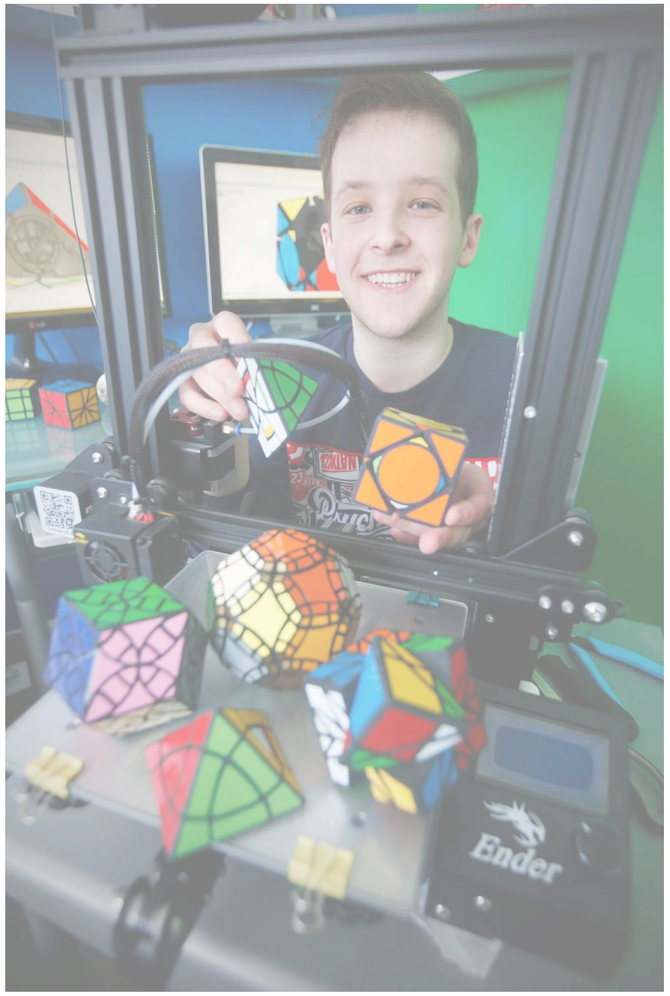
```
(gdb) print set
$1 = boost::unordered_set with 3 elements = {
  [0] = 33,
  [1] = 22,
  [2] = 11
}
```

In this talk

- Addressing the "cons" of debuggers
- Advocating for writing code with tooling and users in mind
- The "how" of Visual Studio Natvis
- The "how" of GDB pretty-printers
- Barely any C++
- Questions at any time!

About me

- Canadian
- Mechanical engineering degree
- I like twisty puzzles
- C++ standards committee (WG21)
- Winnipeg C++ Developers meet up group
- CppNorth volunteer coordinator



Natvis setup

Natvis files

- XML
- `.natvis` file extension

Natvis files

```xml
<?xml version="1.0" encoding="utf-8"?>
<AutoVisualizer xmlns="http://schemas.microsoft.com/vstudio/debugger/natvis/2010">









</AutoVisualizer>
```

Natvis files

```
<?xml version="1.0" encoding="utf-8"?>
<AutoVisualizer xmlns="http://schemas.microsoft.com/vstudio/debugger/natvis/2010">

    <Type Name="MyClass">
        ...
    </Type>




</AutoVisualizer>
```

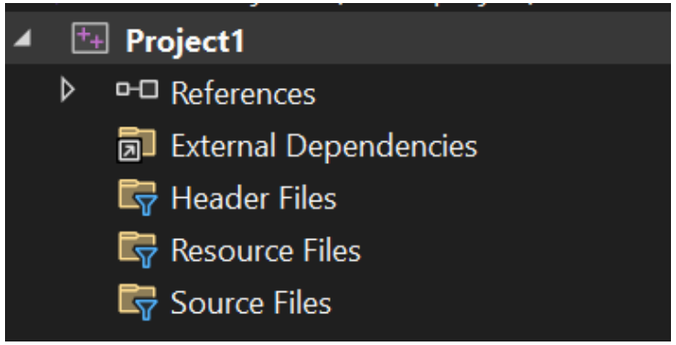

Natvis files

```
<?xml version="1.0" encoding="utf-8"?>
<AutoVisualizer xmlns="http://schemas.microsoft.com/vstudio/debugger/natvis/2010">

    <Type Name="MyClass">
        ...
    </Type>

    <Type Name="MyClassTemplate&lt;*&gt;">
        ...
    </Type>

</AutoVisualizer>
```

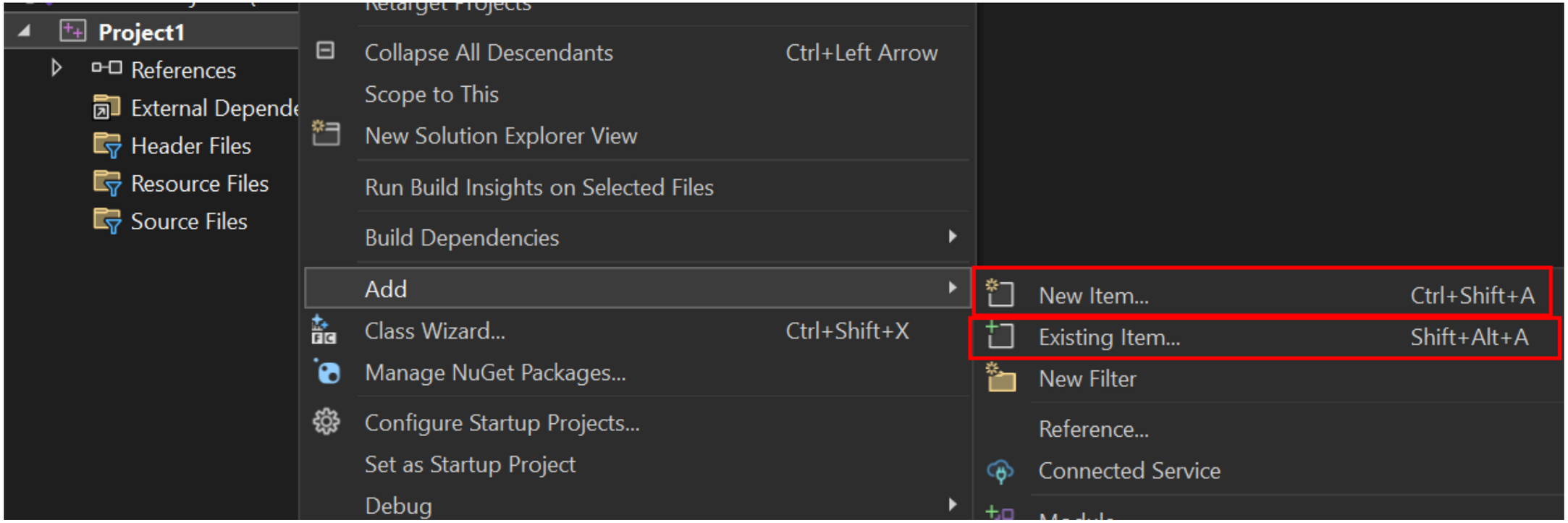
Set up Natvis

- In Visual Studio
- In CMake

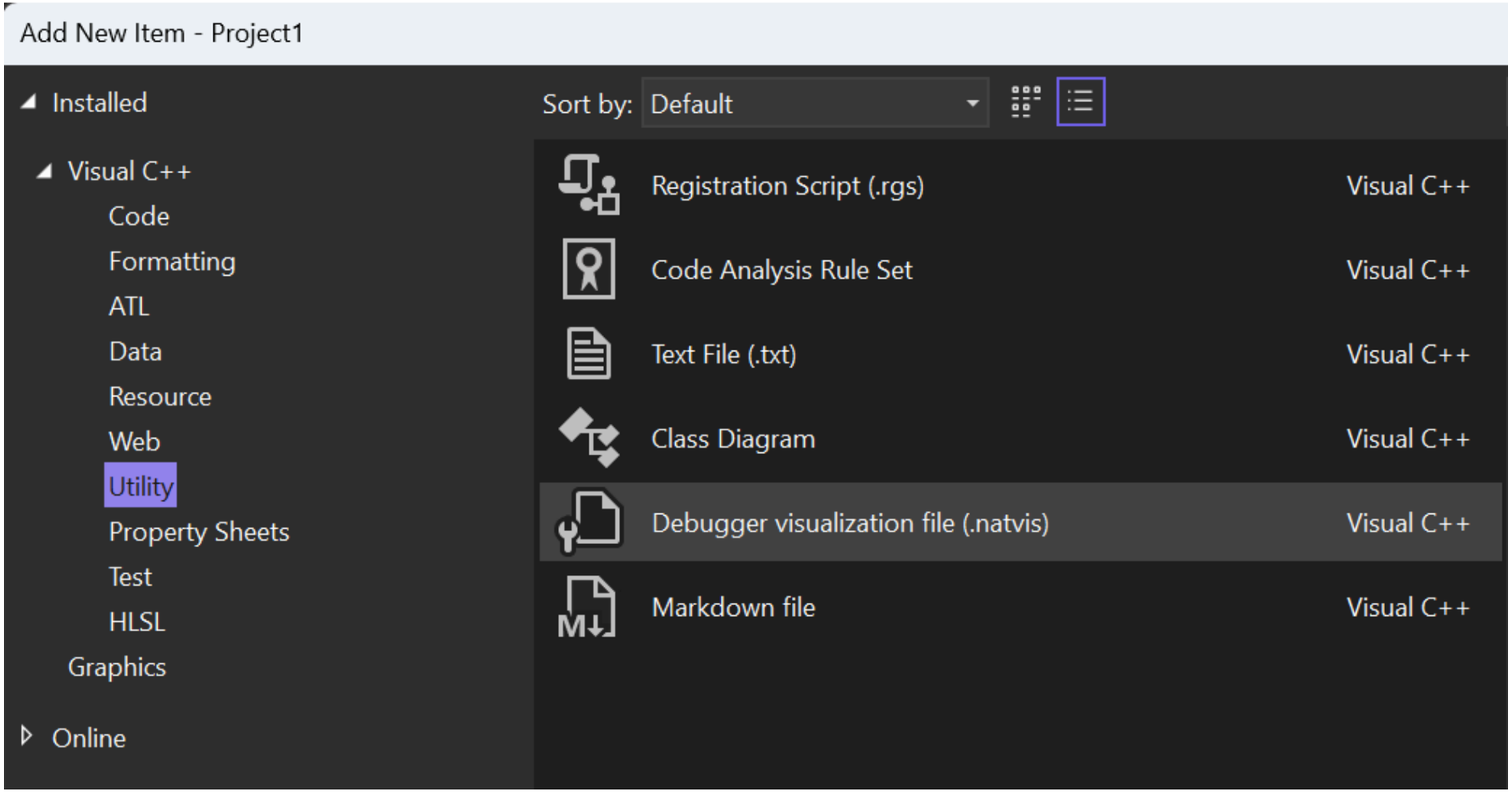
In Visual Studio



In Visual Studio



In Visual Studio



In CMake

```
add_executable(my_target
    # ... all other args
    path/to/my.natvis
)
```

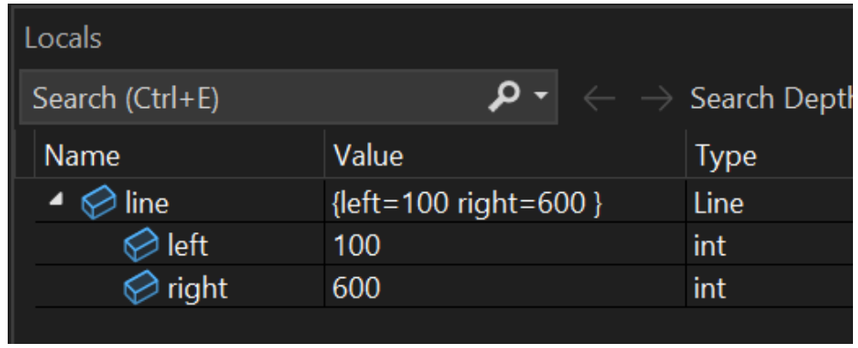
```
target_sources(my_target
    PUBLIC path/to/my.natvis
)
```

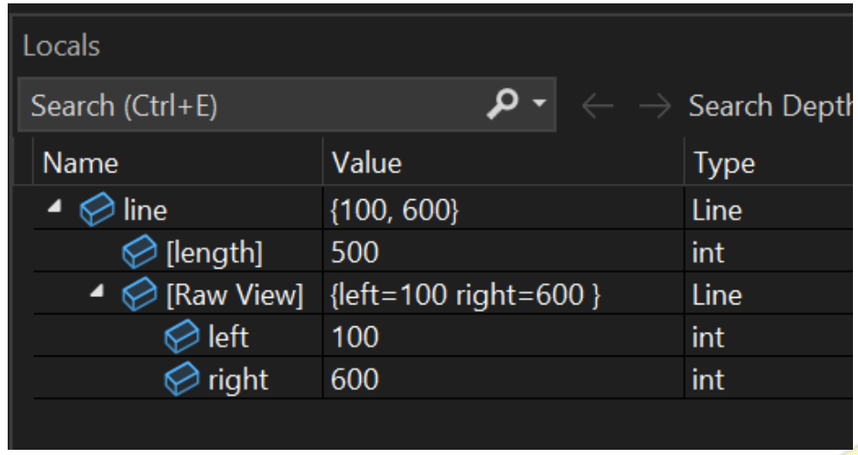
Results

```
struct Line {
    int left;
    int right;
};
```

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand>
        <Item Name="[length]">right - left</Item>
    </Expand>
</Type>
```

```
Line line{ 100, 600 };
```





Writing Natvis

Natvis `<Type>` tag

```
<?xml version="1.0" encoding="utf-8"?>
<AutoVisualizer xmlns="http://schemas.microsoft.com/vstudio/debugger/natvis/2010">

    <Type Name="MyClass">
        ...
    </Type>

    <Type Name="MyClassTemplate&lt;*&gt;">
        ...
    </Type>

</AutoVisualizer>
```

Inside the Natvis `<Type>` tag

- `<AlternativeType>` tag, and attributes on `<Type>`
- `<DisplayString>` tag
- Expansions
- `<Intrinsic>` tag

Importantly

- Any data members, no matter the access specifiers
- *No* user-defined C++ functions
- Some Natvis features are not documented
- This presentation is not exhaustive

`<AlternativeType>` tag

```
<!-- STL.natvis -->
<Type Name="std::unordered_map&lt;*&gt;" Priority="Medium">
    <AlternativeType Name="std::unordered_multimap&lt;*&gt;" />
    <AlternativeType Name="std::unordered_set&lt;*&gt;" />
    <AlternativeType Name="std::unordered_multiset&lt;*&gt;" />
    ...
</Type>
```

`Inheritable` attribute

```
class B {
    // ...
};
```

```
class D : public B {
    // ...
};
```

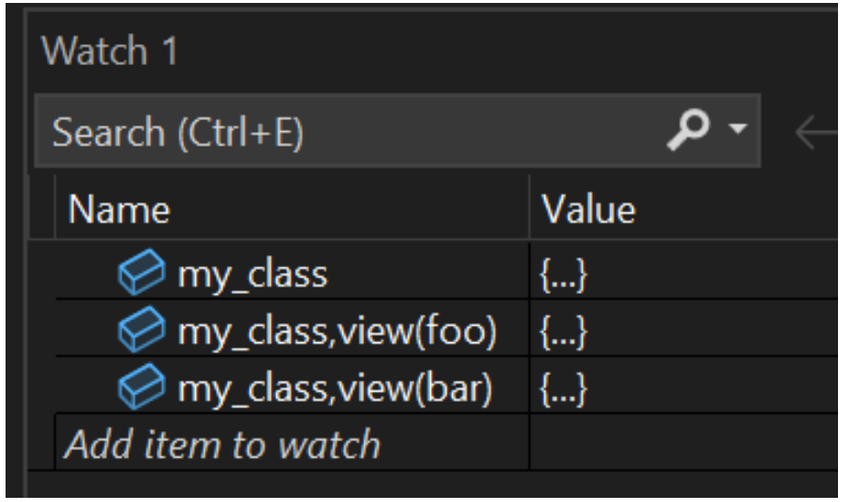
```
<Type Name="B" Inheritable="false">
    ...
</Type>
```

Note: defaults to `"true"`

`IncludeView` and `ExcludeView`

```
<Type Name="MyClass" IncludeView="foo">
    ...
</Type>
```

```
<Type Name="MyClass" ExcludeView="bar">
    ...
</Type>
```



`Priority` attribute

```
<Type Name="MyClass" Priority="Medium">
    ...
</Type>
```

```
<Type Name="MyClass" Priority="MediumHigh">
    ...
</Type>
```

`Priority` with views

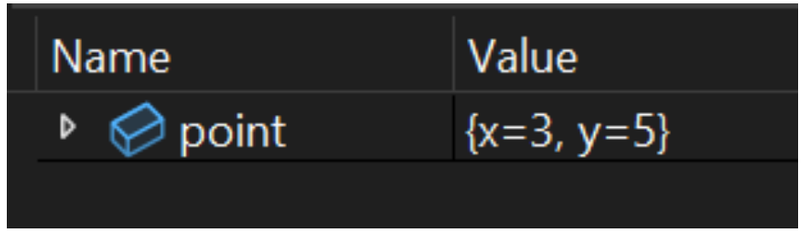
```
<Type Name="std::unordered_map&lt;*&gt;" Priority="Medium">
    ...
</Type>
```

```
<Type Name="std::unordered_map&lt;*&gt;" Priority="MediumHigh"
    ExcludeView="ShowElementsByIndex"
>
    ...
</Type>
```

`<DisplayString>` tag

A format string

```
<Type Name="Point">
    <DisplayString>{{x={x}, y={y}}}</DisplayString>
</Type>
```

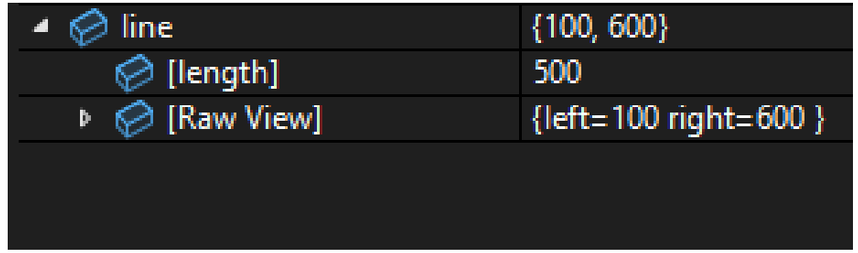


`<DisplayString>` tag

```
<!-- STL.natvis -->
<Type Name="std::optional&lt;*&gt;">
    <DisplayString Condition="!_Has_value">nullopt</DisplayString>
    <DisplayString Condition="_Has_value">{_Value}</DisplayString>
    ...
</Type>
```

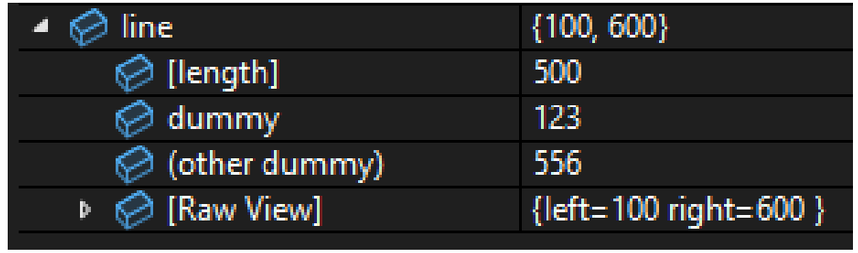
Basic expansions

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand>
        <Item Name="[length]">right - left</Item>


    </Expand>
</Type>
```



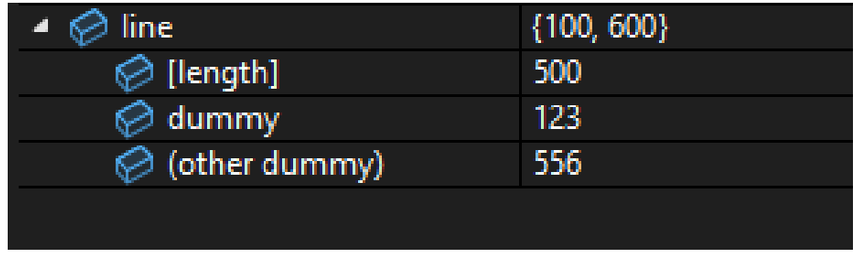
Basic expansions

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand>
        <Item Name="[length]">right - left</Item>
        <Item Name="dummy">123</Item>
        <Item Name="(other dummy)">456 + left</Item>
    </Expand>
</Type>
```



Basic expansions

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">right - left</Item>
        <Item Name="dummy">123</Item>
        <Item Name="(other dummy)">456 + left</Item>
    </Expand>
</Type>
```



`<Item>` tag

```
<Item Name="[length]">right - left</Item>
```

`<Item>` tag

```
<Item
    Name="[length]"


>
    right - left
</Item>
```

`<Item>` tag

```
<Item
    Name="[length]"
    Condition="right - left &gt;= 0"

>
    right - left
</Item>
```

`<Item>` tag

```
<Item
    Name="[length]"
    Condition="right - left &gt;= 0"
    IncludeView="detailed;advanced"
>
    right - left
</Item>
```

`<Item>` tag

```
<Item
    Name="[length]"
    Condition="right - left &gt;= 0"
    ExcludeView="simple"
>
    right - left
</Item>
```

`<ArrayItems>` tag

```
<!-- STL.natvis -->
<Type Name="std::array&lt;*,*&gt;">
    <DisplayString>{{ size={$T2} }}</DisplayString>
    <Expand>



    </Expand>
</Type>
```

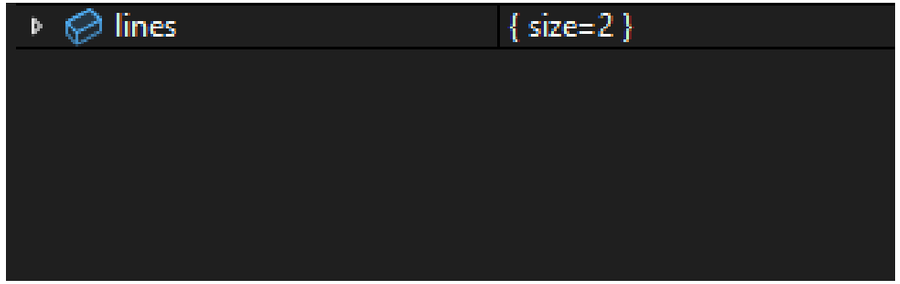
`<ArrayItems>` tag

```
<!-- STL.natvis -->
<Type Name="std::array&lt;*,*&gt;">
    <DisplayString>{{ size={$T2} }}</DisplayString>
    <Expand>
        <ArrayItems>
            <Size>$T2</Size>
            <ValuePointer>_Elems</ValuePointer>
        </ArrayItems>
    </Expand>
</Type>
```

<ArrayItems> tag

```
<Type Name="std::array&lt;*,*&gt;">
    <DisplayString>{{ size={$T2} }}</DisplayString>
    <Expand>
        <ArrayItems>
            <Size>$T2</Size>
            <ValuePointer>_Elems</ValuePointer>
        </ArrayItems>
    </Expand>
</Type>
```

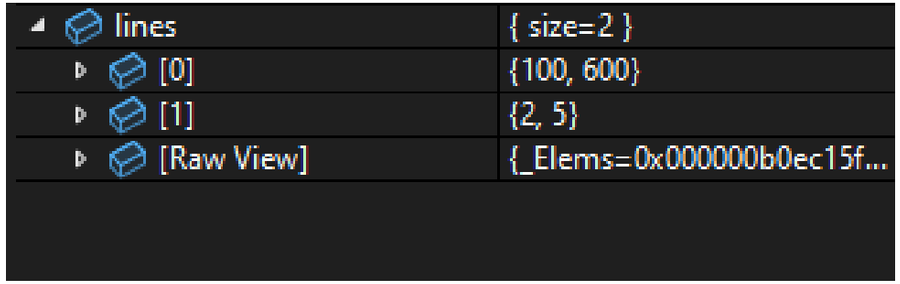
```
std::array<Line, 2> lines{
    Line{100, 600},
    Line{2, 5},
};
```



<ArrayItems> tag

```
<Type Name="std::array&lt;*,*&gt;">
    <DisplayString>{{ size={$T2} }}</DisplayString>
    <Expand>
        <ArrayItems>
            <Size>$T2</Size>
            <ValuePointer>_Elems</ValuePointer>
        </ArrayItems>
    </Expand>
</Type>
```

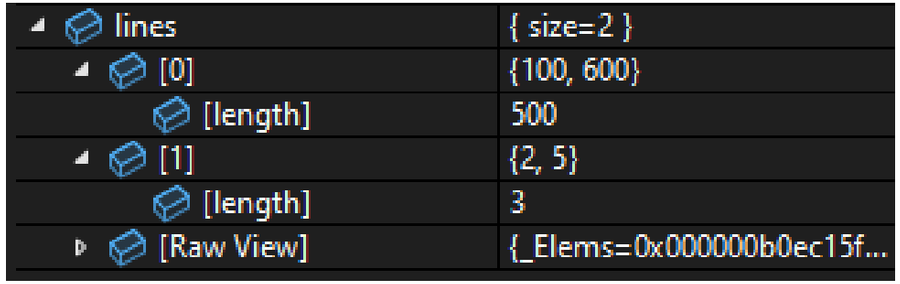
```
std::array<Line, 2> lines{
    Line{100, 600},
    Line{2, 5},
};
```



<ArrayItems> tag

```
<Type Name="std::array&lt;*,*&gt;">
    <DisplayString>{{ size={$T2} }}</DisplayString>
    <Expand>
        <ArrayItems>
            <Size>$T2</Size>
            <ValuePointer>_Elems</ValuePointer>
        </ArrayItems>
    </Expand>
</Type>
```

```
std::array<Line, 2> lines{
    Line{100, 600},
    Line{2, 5},
};
```



Other "automatic" expansions

`<IndexListItems>`

- Expand sequences based on an index

`<LinkedListItems>`

- Expand linked list structures

`<TreeItems>`

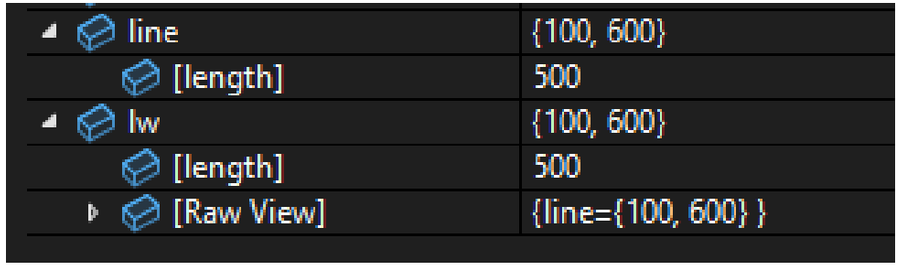
- Expand certain shapes of tree structures

<ExpandedItem> tag

```
struct LineWrapper {
    Line line;
};
```

```
<Type Name="LineWrapper">
    <DisplayString>{line}</DisplayString>
    <Expand>
        <ExpandedItem>line</ExpandedItem>
    </Expand>
</Type>
```

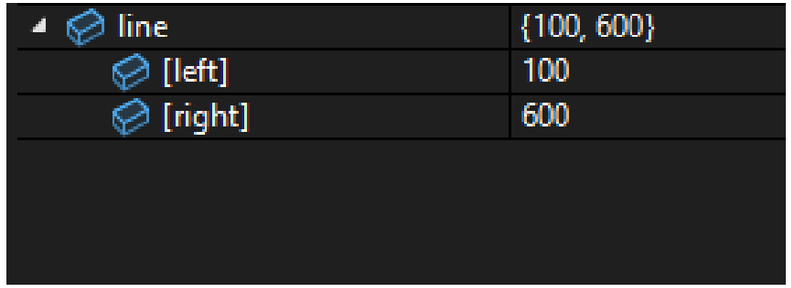
```
Line line{ 100, 600 };
LineWrapper lw{ line };
```



<Synthetic> tag

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[left]">left</Item>
        <Item Name="[right]">right</Item>
    </Expand>
</Type>
```

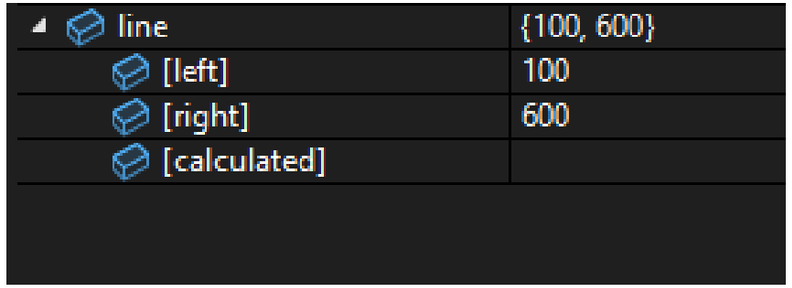
```
Line line{ 100, 600 };
```



<Synthetic> tag

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[left]">left</Item>
        <Item Name="[right]">right</Item>
        <Synthetic Name="[calculated]">




        </Synthetic>
    </Expand>
</Type>
```

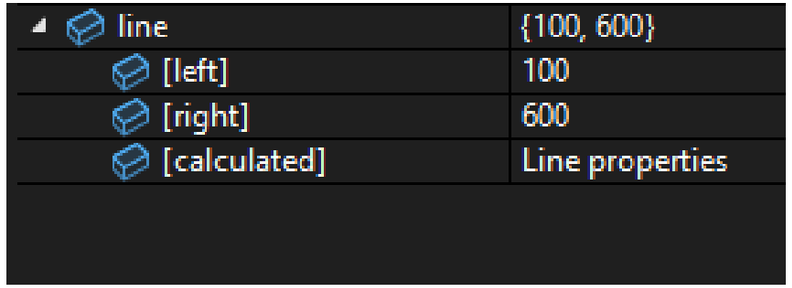
```
Line line{ 100, 600 };
```



<Synthetic> tag

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[left]">left</Item>
        <Item Name="[right]">right</Item>
        <Synthetic Name="[calculated]">
            <DisplayString>Line properties</DisplayString>




        </Synthetic>
    </Expand>
</Type>
```

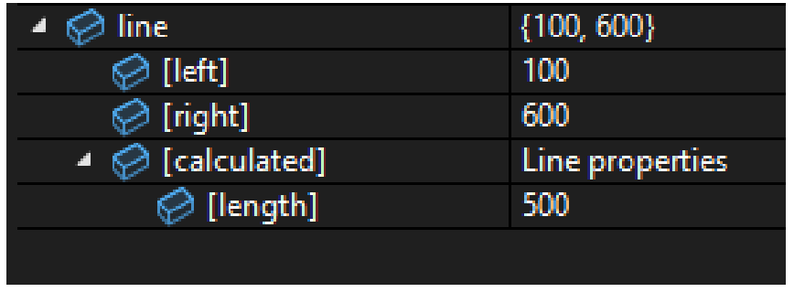
```
Line line{ 100, 600 };
```



<Synthetic> tag

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[left]">left</Item>
        <Item Name="[right]">right</Item>
        <Synthetic Name="[calculated]">
            <DisplayString>Line properties</DisplayString>
            <Expand>
                <Item Name="[length]">right - left</Item>

            </Expand>
        </Synthetic>
    </Expand>
</Type>
```

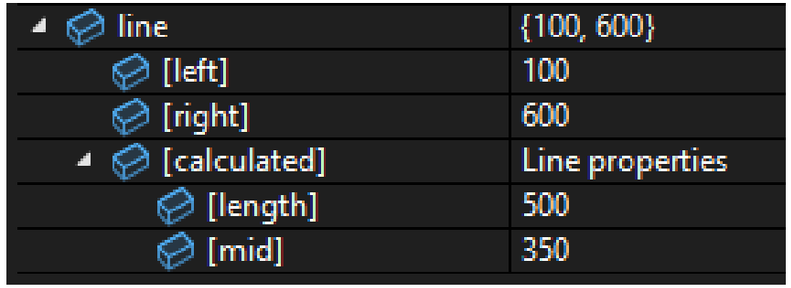
```
Line line{ 100, 600 };
```



`<Synthetic>` tag

```
<Type Name="Line">
    <DisplayString>{{{left}, {right}}}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[left]">left</Item>
        <Item Name="[right]">right</Item>
        <Synthetic Name="[calculated]">
            <DisplayString>Line properties</DisplayString>
            <Expand>
                <Item Name="[length]">right - left</Item>
                <Item Name="[mid]">(left + right)/2</Item>
            </Expand>
        </Synthetic>
    </Expand>
</Type>
```

```
Line line{ 100, 600 };
```

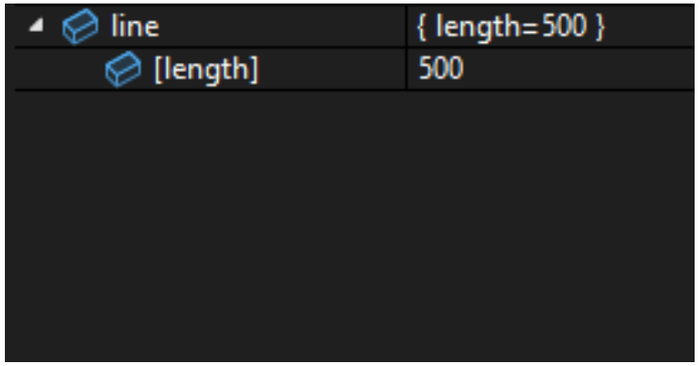


`<CustomListItems>` tag

- A "manual" expansion
- Create variables
- Execute statements
- Branches and loops
- Output items
- Basically a DSL to iterate over your structure

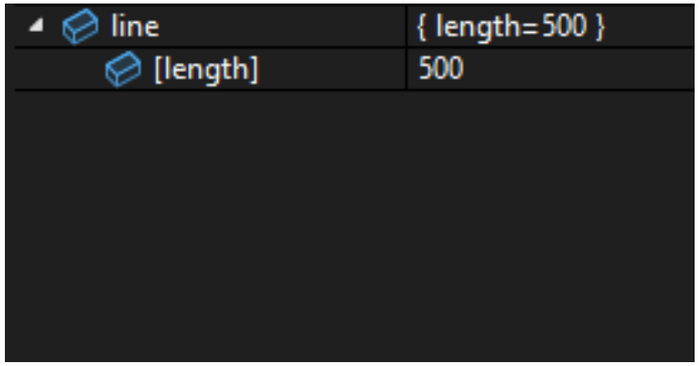
`<CustomListItems>` tag

```
<Type Name="Line">
    <DisplayString>{{ length={right - left} }}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">right - left</Item>
        <CustomListItems>
            ...
        </CustomListItems>
    </Expand>
</Type>
```



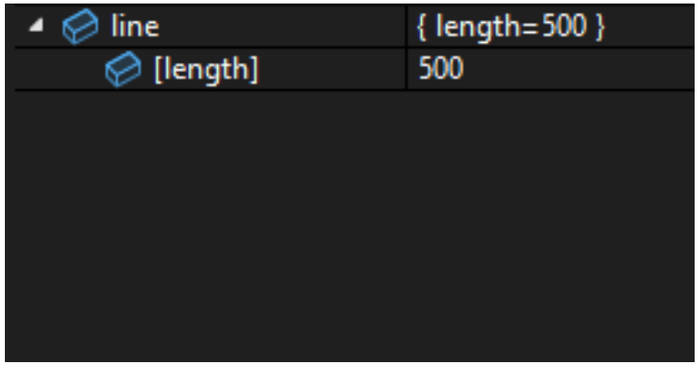
`<CustomListItems>` tag

```
<CustomListItems>




</CustomListItems>
```



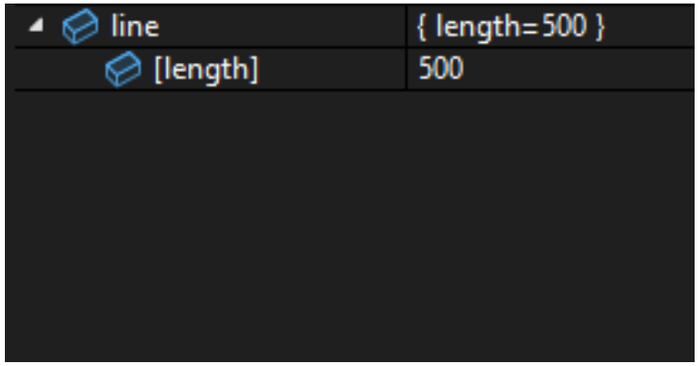
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>




</CustomListItems>
```



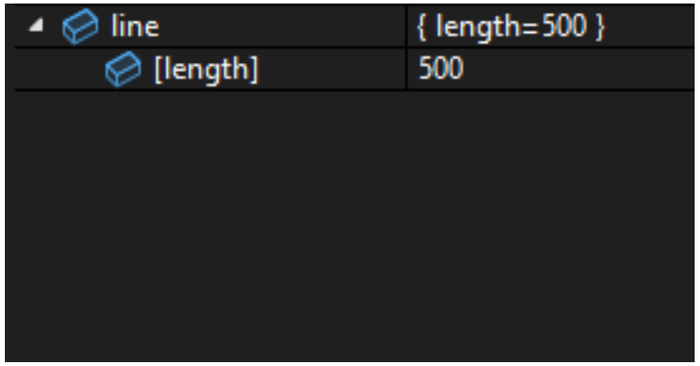
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>









</CustomListItems>
```



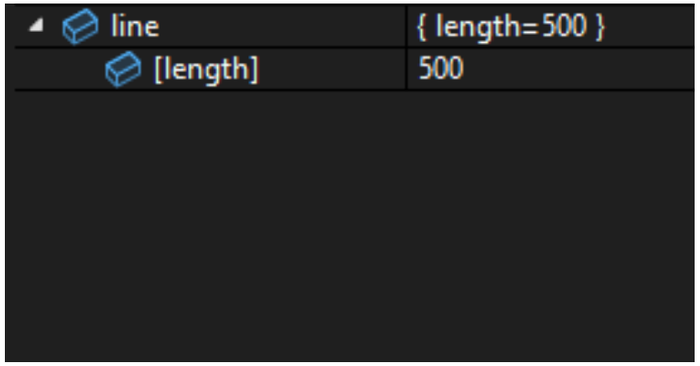
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>









</CustomListItems>
```



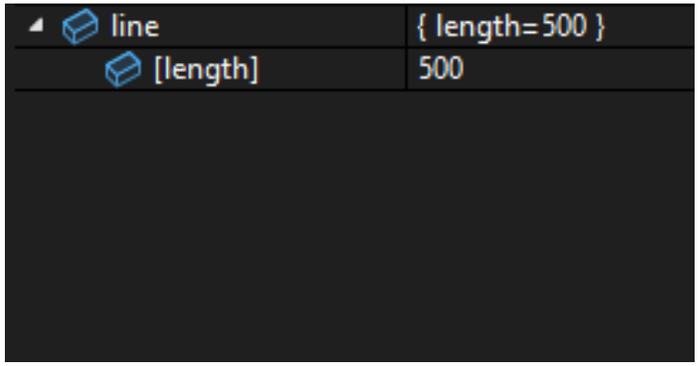
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>









</CustomListItems>
```



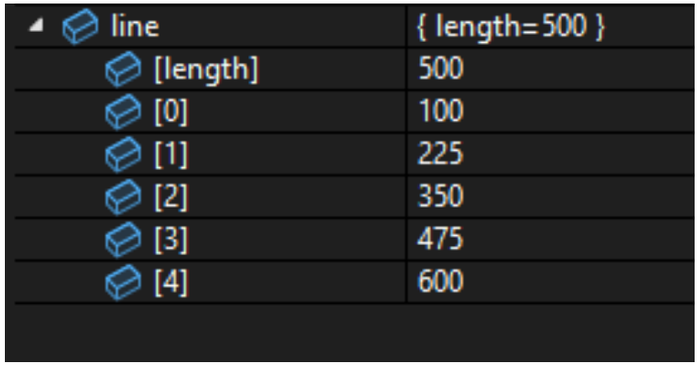
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Loop>

        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>
        <If Condition="current &gt; right">
          <Break/>
        </If>
    </Loop>
</CustomListItems>
```



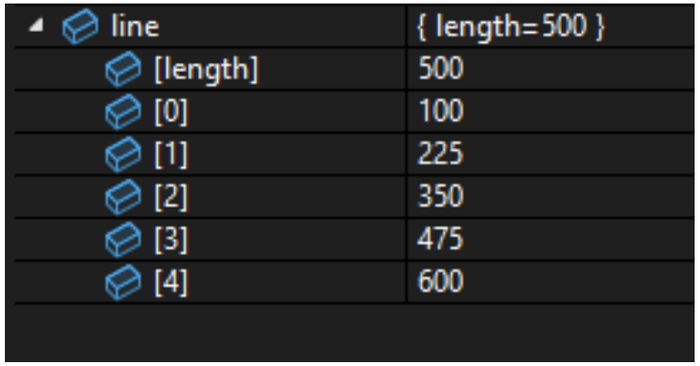
<CustomListItems> tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Loop>
        <Item>current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>
        <If Condition="current &gt; right">
          <Break/>
        </If>
    </Loop>
</CustomListItems>
```



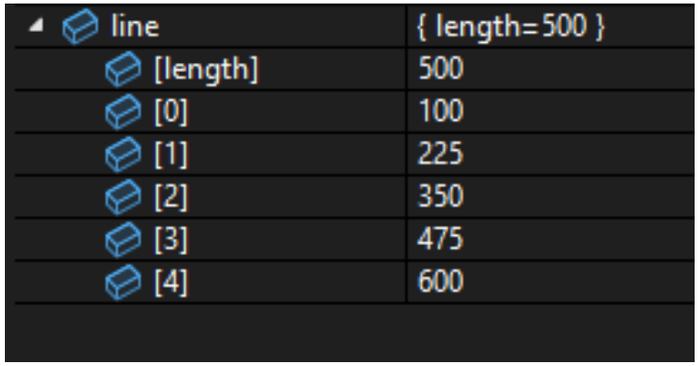
`<CustomListItems>` tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Loop>
        <Item>current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>

        <Break Condition="current &gt; right"/>

    </Loop>
</CustomListItems>
```



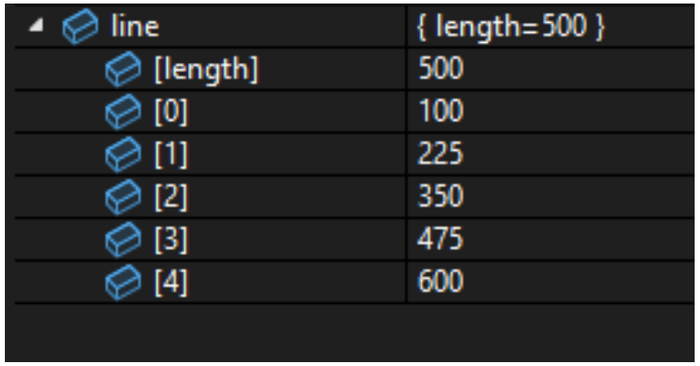
<CustomListItems> tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Loop Condition="current &lt;= right">
        <Item>current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>



    </Loop>
</CustomListItems>
```



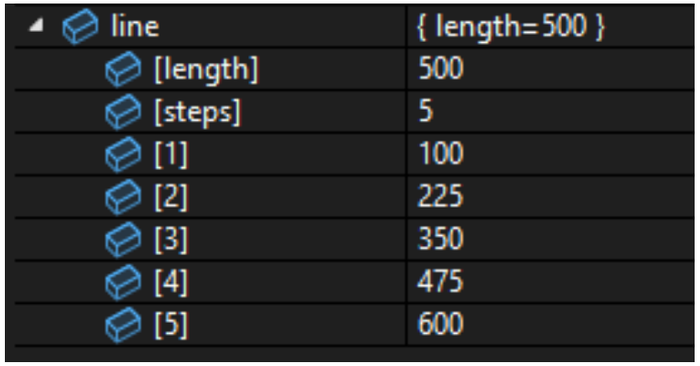
<CustomListItems> tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>


    <Loop Condition="current &lt;= right">
        <Item>current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>
    </Loop>
</CustomListItems>
```



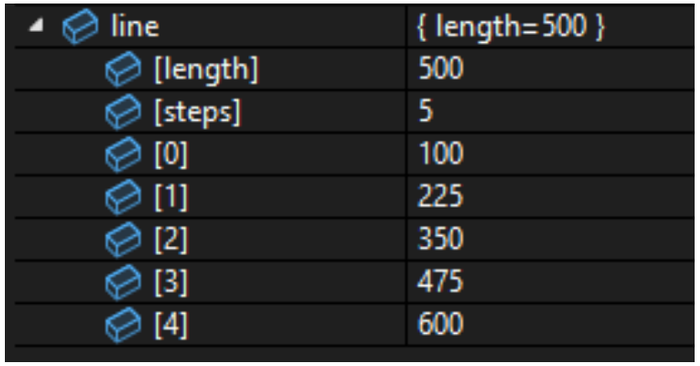
<CustomListItems> tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Item Name="[steps]">steps</Item>
    <Loop Condition="current &lt;= right">
        <Item>current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>
    </Loop>
</CustomListItems>
```



<CustomListItems> tag

```
<CustomListItems>
    <Variable Name="steps" InitialValue="5"/>
    <Variable Name="step_size"
        InitialValue="(right - left) / (steps - 1)"/>
    <Variable Name="current" InitialValue="left"/>
    <Variable Name="index" InitialValue="0"/>

    <Item Name="[steps]">steps</Item>
    <Loop Condition="current &lt;= right">
        <Item Name="[{index}]">current</Item>
        <Exec>++index</Exec>
        <Exec>current = left + index * step_size</Exec>
    </Loop>
</CustomListItems>
```



`<Intrinsic>` tag

- User-defined functions within Natvis
- Member function if within `<Type>` tag
- Free function if outside `<Type>` tags
- Can have parameters
- Can be overloaded

<Intrinsic> tag

```
<Type Name="Line">

    <DisplayString>{{ length={right - left} }}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">right - left</Item>
        <CustomListItems>
            <Variable Name="steps" InitialValue="5"/>
            <Variable Name="step_size"
                InitialValue="(right - left) / (steps - 1)"/>
            <Variable Name="current" InitialValue="left"/>
            <Variable Name="index" InitialValue="0"/>
            ...
        </CustomListItems>
    </Expand>
</Type>
```

`<Intrinsic>` tag

```
<Type Name="Line">
    <Intrinsic Name="length"                                    />
    <DisplayString>{{ length={right - left} }}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">right - left</Item>
        <CustomListItems>
            <Variable Name="steps" InitialValue="5"/>
            <Variable Name="step_size"
                InitialValue="(right - left) / (steps - 1)"/>
            <Variable Name="current" InitialValue="left"/>
            <Variable Name="index" InitialValue="0"/>
            ...
        </CustomListItems>
    </Expand>
</Type>
```

<Intrinsic> tag

```
<Type Name="Line">
    <Intrinsic Name="length" Expression="right - left"/>
    <DisplayString>{{ length={right - left} }}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">right - left</Item>
        <CustomListItems>
            <Variable Name="steps" InitialValue="5"/>
            <Variable Name="step_size"
                InitialValue="(right - left) / (steps - 1)"/>
            <Variable Name="current" InitialValue="left"/>
            <Variable Name="index" InitialValue="0"/>
            ...
        </CustomListItems>
    </Expand>
</Type>
```

<Intrinsic> tag

```
<Type Name="Line">
    <Intrinsic Name="length" Expression="right - left"/>
    <DisplayString>{{ length={length()} }}</DisplayString>
    <Expand HideRawView="true">
        <Item Name="[length]">length()</Item>
        <CustomListItems>
            <Variable Name="steps" InitialValue="5"/>
            <Variable Name="step_size"
                InitialValue="length() / (steps - 1)"/>
            <Variable Name="current" InitialValue="left"/>
            <Variable Name="index" InitialValue="0"/>
            ...
        </CustomListItems>
    </Expand>
</Type>
```

Official docs

"Create custom views of C++ objects in the debugger using the Natvis framework"

- **https://learn.microsoft.com/en-us/visualstudio/debugger/create-custom-views-of-native-objects**

"Implement NatVis custom intrinsic function for C++"

- **https://learn.microsoft.com/en-us/visualstudio/debugger/implementing-natvis-intrinsic-function**
- **New article from January 2025**

My articles

"Natvis for boost::unordered_map, and how to use <Intrinsic> elements"

- https://blog.ganets.ky/NatvisForUnordered/

"Natvis for boost::concurrent_flat_map, and why fancy pointers are hard"

- https://blog.ganets.ky/NatvisForUnordered2/

"I want to write automated Natvis testing"

- https://blog.ganets.ky/NatvisTesting/

For a subsequent talk

- `Optional` attribute for SFINAE-like features
- Navigating advanced `<Intrinsic>`
- Writing Natvis to allow user-injected behaviour
- Attempts to automate testing of Natvis
- My Natvis wish list

GDB pretty-printer setup

GDB pretty-printer setup

- Basic GDB
- .gdbinit file
- Linking custom pretty-printers
- Plumbing the pretty-printers

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
> clang++ main.cpp
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Type "apropos word" to search for commands related to "word".
(gdb)
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
> clang++ main.cpp
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Type "apropos word" to search for commands related to "word".
(gdb) file a.out
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
> clang++ main.cpp
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Type "apropos word" to search for commands related to "word".
(gdb) file a.out
Reading symbols from a.out...
(No debugging symbols found in a.out)
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
> clang++ main.cpp -g
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Type "apropos word" to search for commands related to "word".
(gdb) file a.out
Reading symbols from a.out...
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
$1 = 0
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
$1 = 0
(gdb) next
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
$1 = 0
(gdb) next
5       }
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
$1 = 0
(gdb) next
5       }
(gdb) print z
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) start
Temporary breakpoint 1 at 0x1134: file main.cpp, line 2.
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Temporary breakpoint 1, main () at main.cpp:2
2         int x = 5;
(gdb) next
3         int y = 6;
(gdb) next
4         int z = x + y;
(gdb) print z
$1 = 0
(gdb) next
5       }
(gdb) print z
$2 = 11
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) file a.out
Reading symbols from a.out...
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) file a.out
Reading symbols from a.out...
(gdb) break 5
Breakpoint 1 at 0x114b: file main.cpp, line 5.
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) file a.out
Reading symbols from a.out...
(gdb) break 5
Breakpoint 1 at 0x114b: file main.cpp, line 5.
(gdb) run
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:5
5       }
```

Basic GDB

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
(gdb) file a.out
Reading symbols from a.out...
(gdb) break 5
Breakpoint 1 at 0x114b: file main.cpp, line 5.
(gdb) run
Starting program: path/to/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:5
5       }
(gdb) print z
$1 = 11
```

.gdbinit file

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
# .gdbinit
file a.out
break 5
run
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
warning: File "path/to/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "...".
To enable execution of this file add
        add-auto-load-safe-path path/to/.gdbinit
line to your configuration file
"/home/braden/.config/gdb/gdbinit".
...etc...
```

.gdbinit file

```
# /home/braden/.config/gdb/gdbinit
add-auto-load-safe-path path/to/.gdbinit
```

.gdbinit file

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
# .gdbinit
file a.out
break 5
run
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Breakpoint 1 at 0x114b: file main.cpp, line 5.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:5
5       }
```

.gdbinit file

```
0 // main.cpp
1 int main() {
2     int x = 5;
3     int y = 6;
4     int z = x + y;
5 }
```

```
# .gdbinit
file a.out
break 5
run
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Breakpoint 1 at 0x114b: file main.cpp, line 5.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "path/to/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:5
5       }
(gdb) print z
$1 = 11
```

Linking custom pretty-printers

```
0 // main.cpp
1 class MyType {
2     // ...
3 };
4 int main() {
5     MyType my_type{};
6 }
```

```
# MyTypePrinter.py
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"

# ...etc for the plumbing...
```

```
# .gdbinit
set print pretty on
source MyTypePrinter.py
file a.out
break 6
run
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Breakpoint 1, main () at main.cpp:6
6       }
```

Linking custom pretty-printers

```
0 // main.cpp
1 class MyType {
2     // ...
3 };
4 int main() {
5     MyType my_type{};
6 }
```

```
# MyTypePrinter.py
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"

# ...etc for the plumbing...
```

```
# .gdbinit
set print pretty on
source MyTypePrinter.py
file a.out
break 6
run
```

```
> gdb
```

```
GNU gdb (GDB) 14.0.50.20221220-git
Copyright (C) 2022 Free Software Foundation, Inc.
...etc...
Breakpoint 1, main () at main.cpp:6
6       }
(gdb) print my_type
$1 = Hello, printer!
```

Plumbing the pretty-printers

```
import gdb.printing

class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"

def build_pretty_printer():
```

Plumbing the pretty-printers

```
import gdb.printing
```

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
```

Plumbing the pretty-printers

```
import gdb.printing
```

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('MyType', '^MyType$', MyPrinter)
```

Plumbing the pretty-printers

```
import gdb.printing
```

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('MyType', '^MyType$', MyPrinter)
    pp.add_printer('MyTemplateType', '^MyTemplateType<.*>$', MyPrinter)
```

Plumbing the pretty-printers

```
import gdb.printing
```

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('MyType', '^MyType$', MyPrinter)
    pp.add_printer('MyTemplateType', '^MyTemplateType<.*>$', MyPrinter)
    return pp
```

```
gdb.printing.register_pretty_printer(


)
```

Plumbing the pretty-printers

```
import gdb.printing
```

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('MyType', '^MyType$', MyPrinter)
    pp.add_printer('MyTemplateType', '^MyTemplateType<.*>$', MyPrinter)
    return pp
```

```
gdb.printing.register_pretty_printer(
    gdb.current_objfile(),
    build_pretty_printer()
)
```

Writing GDB pretty-printers

GDB pretty-printers

- Python
- Therefore, full power of a programming language
- E.g. external libraries
- Easier to write than Natvis (in my opinion)

Writing GDB pretty-printers

- Constructor and `to_string()` method
- `children()` method
- Using GDB from VSCode

Constructor and `to_string()`

```
class MyTypePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('MyType', '^MyType$', MyPrinter)
    pp.add_printer('MyTemplateType', '^MyTemplateType<.*>$', MyPrinter)
    return pp
```

```
gdb.printing.register_pretty_printer(
    gdb.current_objfile(),
    build_pretty_printer()
)
```

```
struct Line {
    int left;
    int right;
};
```

Constructor and `to_string()`

```
class LinePrinter:
    def __init__(self, val):
        pass
    def to_string(self):
        return "Hello, printer!"
```

```
def build_pretty_printer():
    pp = gdb.printing.RegexpCollectionPrettyPrinter("my_printers")
    pp.add_printer('Line', '^Line$', LinePrinter)

    return pp
```

```
gdb.printing.register_pretty_printer(
    gdb.current_objfile(),
    build_pretty_printer()
)
```

```
struct Line {
    int left;
    int right;
};
```

Constructor

```
class LinePrinter:
    def __init__(self, val):
        pass

    def to_string(self):
        return "Hello, printer!"
```

Constructor

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        return "Hello, printer!"
```

`to_string()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        return "Hello, printer!"
```

`to_string()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
```

`to_string()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
        right = self.val["right"]
```

`to_string()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
        right = self.val["right"]
        return f"{{{left}, {right}}}"
```

`to_string()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
        right = self.val["right"]
        return f"{{{left}, {right}}}"
```

```
Line line{ 100, 600 };
```

```
(gdb) print line
$1 = {100, 600}
```

`children()` method

- Must return a Python iterator
- Each item is a name-value pair

`children()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
        right = self.val["right"]
        length = right - left
        return f"Line of length {length}"
```

`children()` method

```
class LinePrinter:
    def __init__(self, val):
        self.val = val

    def to_string(self):
        left = self.val["left"]
        right = self.val["right"]
        length = right - left
        return f"Line of length {length}"

    def children(self):
        def generator():
            # ...
        return generator()
```


`children()` method

```
def children(self):
    def generator():









    return generator()
```

`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]









    return generator()
```

`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0









    return generator()
```


`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0

        yield "steps", steps









    return generator()
```


`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0

        yield "steps", steps

        while current <= right:




    return generator()
```

`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0

        yield "steps", steps

        while current <= right:
            yield f"[{index}]", current



    return generator()
```


`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0

        yield "steps", steps

        while current <= right:
            yield f"[{index}]", current
            index += 1
            current = left + index * step_size

    return generator()
```

`children()` method

```
def children(self):
    def generator():
        left = self.val["left"]
        right = self.val["right"]

        steps = 5
        step_size = (right - left) / (steps - 1)
        current = left
        index = 0

        yield "steps", steps

        while current <= right:
            yield f"[{index}]", current
            index += 1
            current = left + index * step_size

    return generator()
```

```
Line line{ 100, 600 };
```

```
(gdb) print line
$1 = Line of length 500 = {
  steps = 5,
  [0] = 100,
  [1] = 225,
  [2] = 350,
  [3] = 475,
  [4] = 600
}
```

Using GDB from VSCode

- Take advantage of VSCode's GUI
- Cannot use ".gdbinit" file
- Requires GDB 14.1 otherwise `to_string()` is broken
- (https://sourceware.org/bugzilla/show_bug.cgi?id=11335)

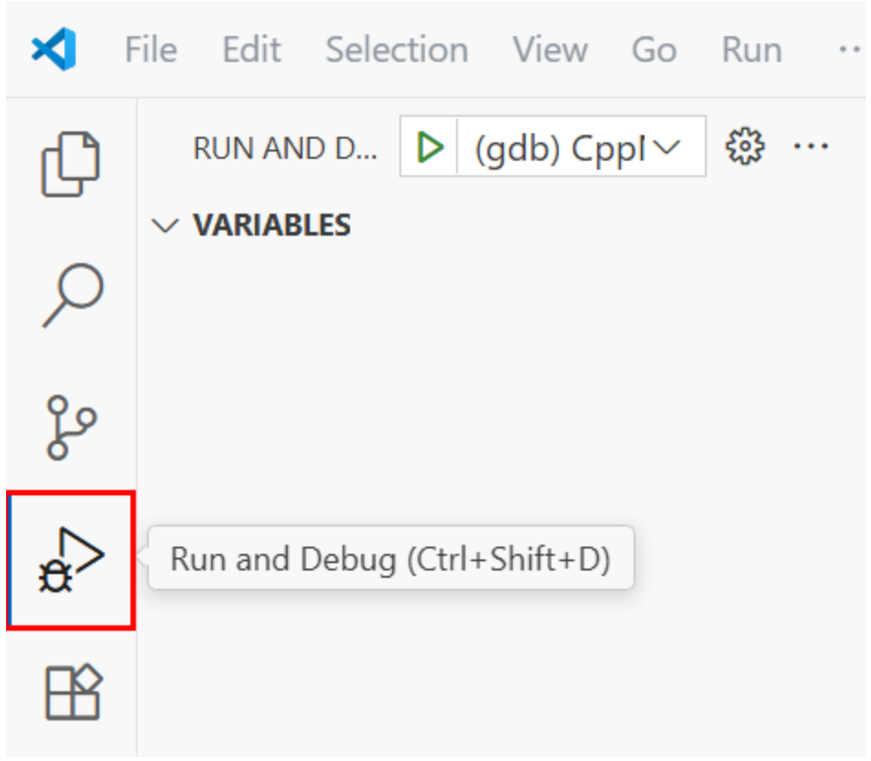
Using GDB from VSCode

```
// .vscode/launch.json
{
    "version": "0.2.0",
    "configurations": [
        {
            // ...



        }
    ]
}
```

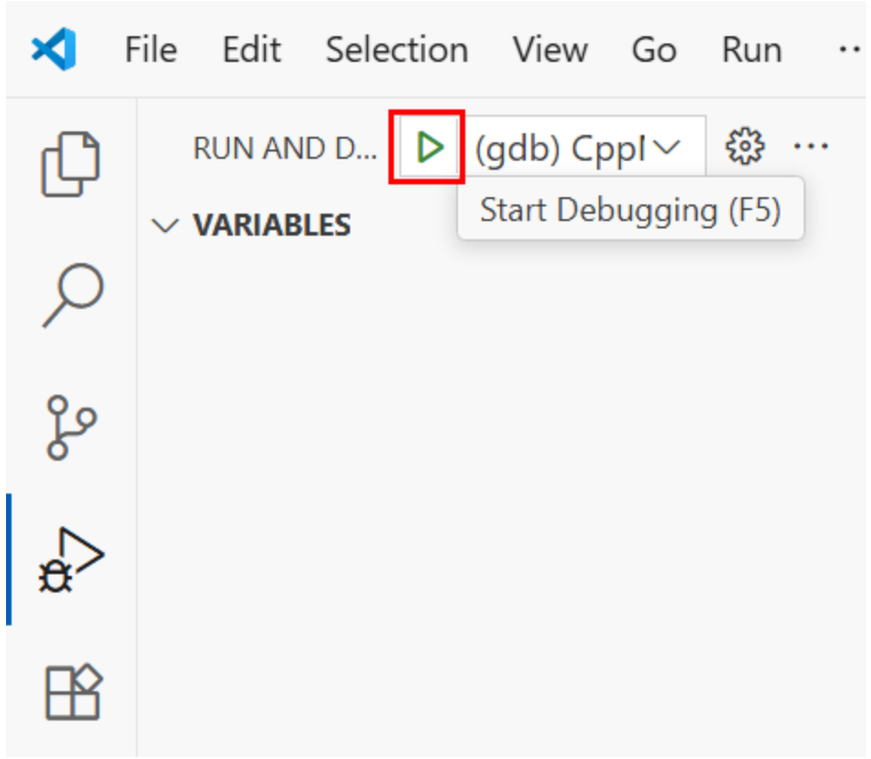
Using GDB from VSCode

```
// .vscode/launch.json
{
    "version": "0.2.0",
    "configurations": [
        {
            // ...
            "setupCommands": [
                // ...
            ]
        }
    ]
}
```

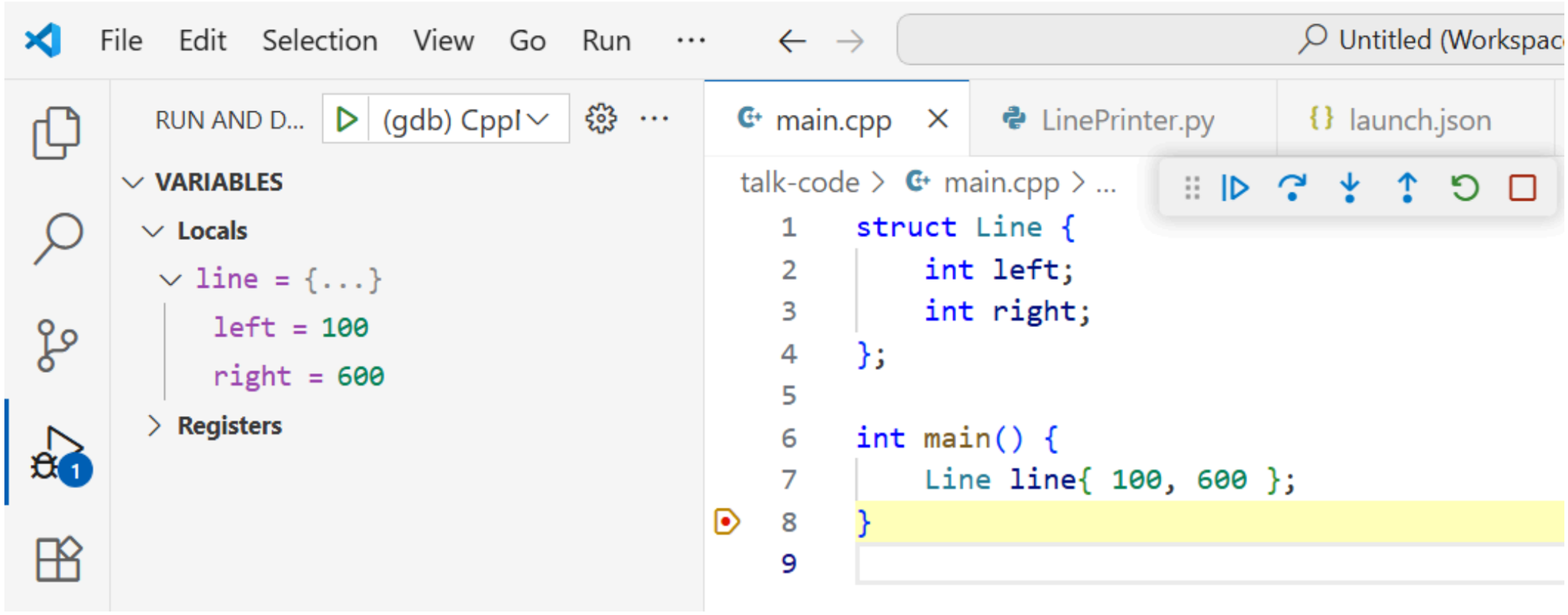
Using GDB from VSCode



Using GDB from VSCode



Using GDB from VSCode



Using GDB from VSCode

```
"setupCommands": [




]
```

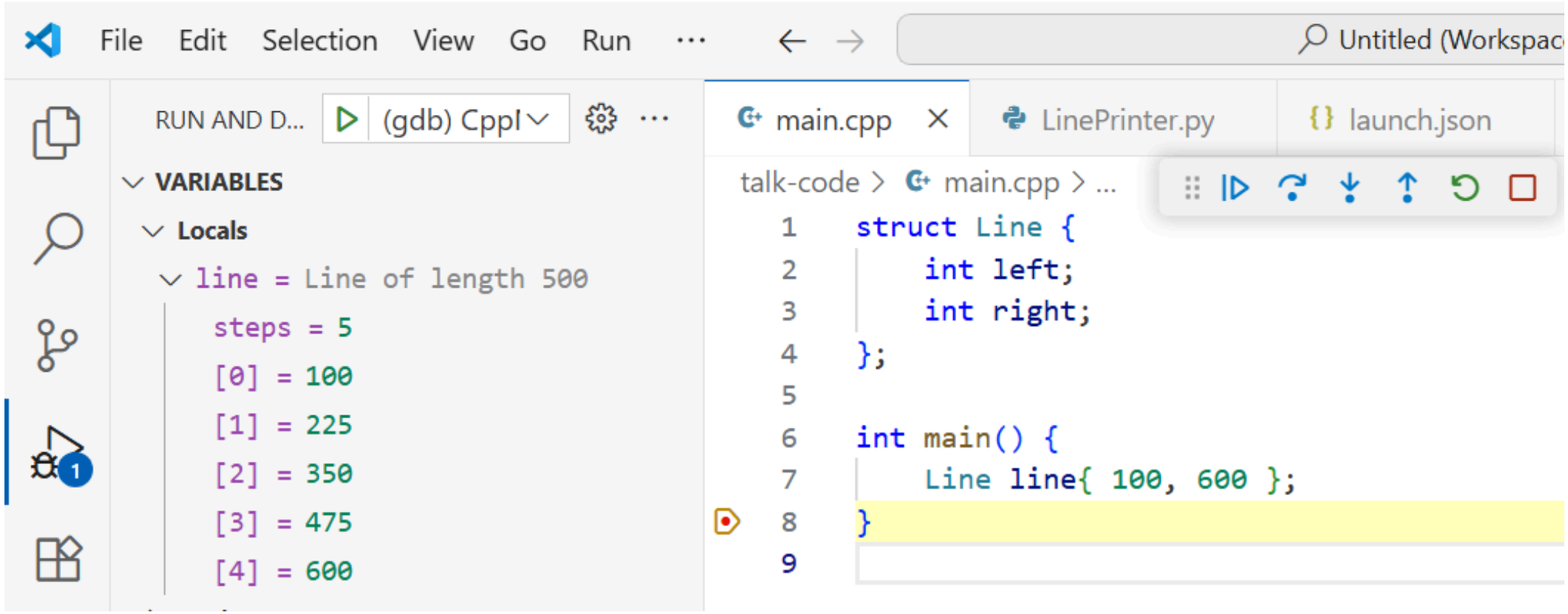
Using GDB from VSCode

```
"setupCommands": [
    {
        "description": "Enable pretty-printing for gdb",
        "text": "-enable-pretty-printing",
        "ignoreFailures": true
    },




]
```

Using GDB from VSCode

```
"setupCommands": [
    {
        "description": "Enable pretty-printing for gdb",
        "text": "-enable-pretty-printing",
        "ignoreFailures": true
    },
    {
        "description": "Load pretty printers",
        "text": "source ${workspaceFolder}/LinePrinter.py",
        "ignoreFailures": true
    }
]
```

Using GDB from VSCode



Resources

"Python API"

- https://sourceware.org/gdb/current/onlinedocs/gdb.html/Python-API.html

"Visualizing boost::unordered_map in GDB, with pretty-printer customization points"

- https://blog.ganets.ky/PrettyPrinter/

For a subsequent talk

- Go into `gdb.Value` and `gdb.Type`
- Writing pretty-printers to allow user-injected behaviour
- Xmethods
- Embedding pretty-printers in the binary itself

Conclusion

Write visualizers

- Now you know how!
- Writing a visualizer is accessible to you!
- Colleagues will thank you
- Your future self will thank you

Ultimately I want

- Developer empowerment
- Removal of barriers to understanding
- Libraries that are more accessible to use
- Widely consumed libraries to have debugger support

Our work is about more than the source code

Our work is about people

Let's make our code more accessible

Thank you!

Braden Ganetsky

braden@ganets.ky
GitHub @k3DW