Runtime-configurable data correlation

A C++-template-based framework for the automatic generation of data accessors with a focus on low programmer effort and low code repetition

Birk Magnussen
FreeSpace 2 Source Code Project
birk@bmagnu.net

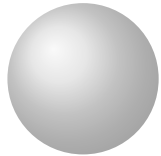
About Me

- Received a Dr. rer. nat. in computer science in 2024 from the University of Kassel
- Specializes in optical sensor data processing using machine learning
- Member of the FreeSpace 2 Source Code Project since 2020
- Core contributions in architecture and API design, rendering, and VR support

Programmer-Defined Correlation In-/Outputs:

Properties
(inputs):

radius
velocity
...



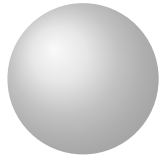
Render settings
(outputs):

scale
color tint
...

Programmer-Defined Correlation In-/Outputs:

Properties (inputs):

radius
velocity
...



Render settings (outputs):

scale
color tint
...

User-Defined Correlations at Runtime:

radius
velocity
...

multiply by →

scale
color tint
...

"Just hardcode it!"

1. Lots of boilerplate code for parsing at runtime
2. The 3D-rendering example has more than just spheres
   - → Other object types have different in- and outputs
   - → We want to avoid repeating boilerplate code
   - → *Some* inputs may be shared between different object types, we want to avoid repeating code for these common inputs
3. A large number of possible inputs bloats the code


```
Object Sphere:
    ...
    var Radius
    var Velocity
    ...
    var Correlations:
        with Inputs:
            Radius
            Velocity
        with Outputs:
            Scale
            Color Tint
```

```
func render:
    var Sphere = ...
    var Scale = Sphere.Correlations
        :GetOutput(Scale)
    renderWithScale(Sphere, Scale)
```


```
Object Sphere:
    ...
    var R
    var V
    ...
    var C                                          elations
        w
                                                   Scale)
        w
            Scale
            Color Tint
```

Goal

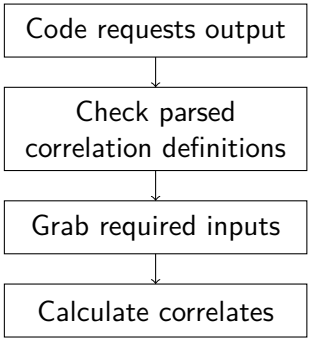
Make `Correlations` a reusable library.

Don't require more information and code for each inputs than the member variable name.

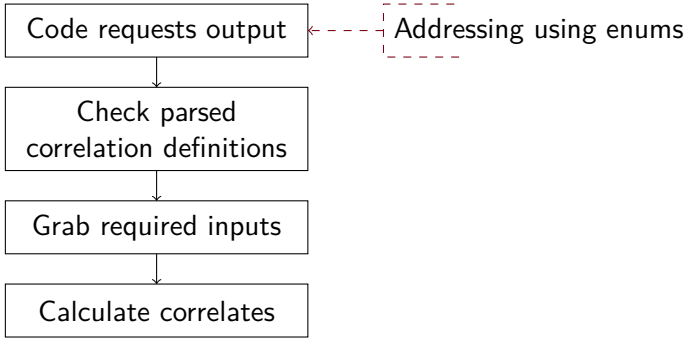
Automatically generate accessor code at compile-time.

→ Possible in C++ with template metaprogramming

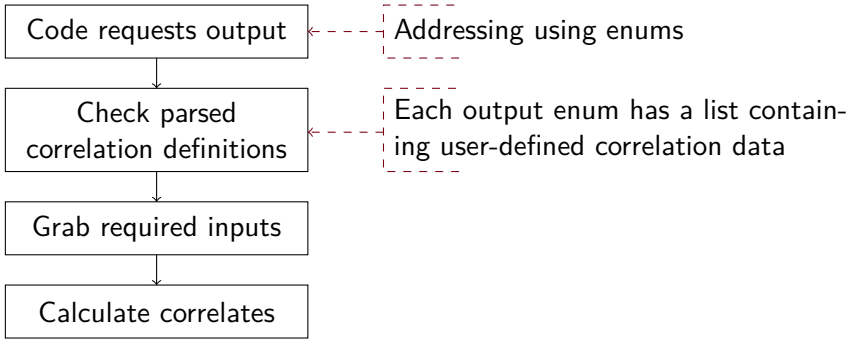
Solution Concept



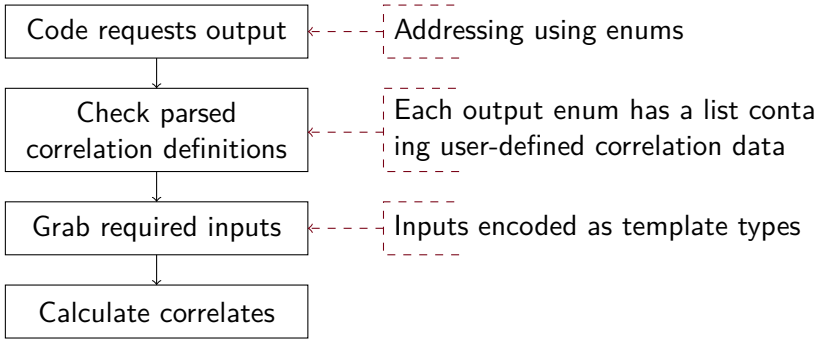
Solution Concept



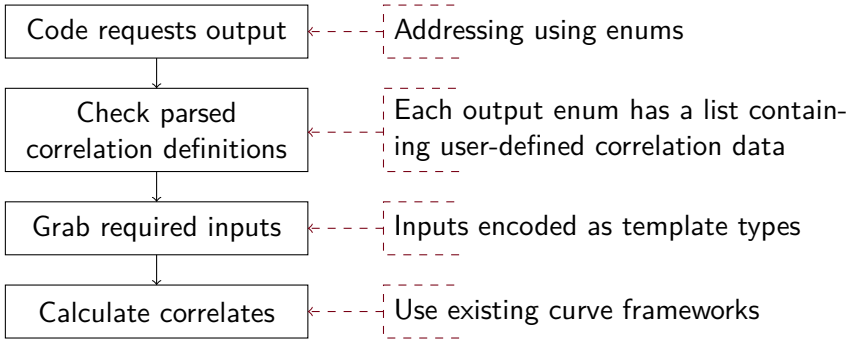
Solution Concept



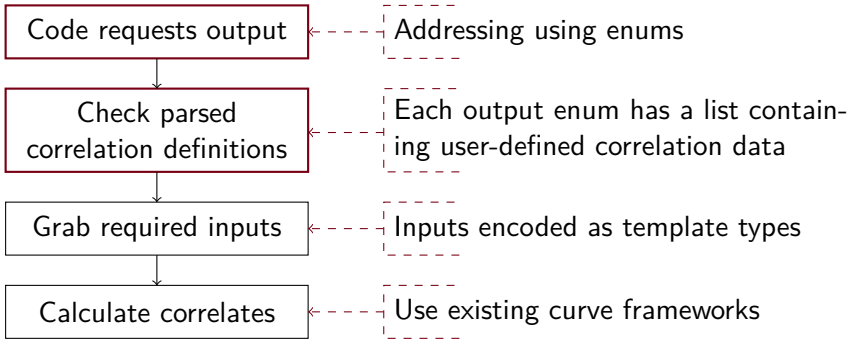
Solution Concept



Solution Concept



Solution Concept



Basic Template Design

The Main Struct


```
1  enum class SphereRenderOutputs { Radius, Velocity, MAX };
2
3  template<typename OutputEnum, typename InputType,
4      /* TODO Inputs */>
5      struct Correlations {
6          float getValue(InputType in, OutputEnum out);
7      };
```

Note: This is simplified, in the code, there is an additional struct type that holds data and strings required for parsing.

Basic Template Design

The Main Struct


```
1  enum class SphereRenderOutputs { Radius, Velocity, MAX };
2
3  template<typename OutputEnum, typename InputType
4      typename InputGrab...>
5      struct Correlations {
6          using InputGrabTuple = tuple<InputGrab...>;
7          array<ParseData, OutputEnum::MAX> userDefs;
8          float getValue(InputType in, OutputEnum out);
9      };
```

Note: This is simplified, in the code, there is an additional struct type that holds data and strings required for parsing.

Basic Template Design

The Main Struct


```
1  enum class SphereRenderOutputs { Radius, Velocity, MAX };
2
3  template<typename OutputEnum, typename InputType
4      typename InputGrab...>
5      struct Correlations {
6          using InputGrabTuple = tuple<InputGrab...>;
7          array<ParseData, OutputEnum::MAX> userDefs;
8          float getValue(InputType in, OutputEnum out);
9      };
```

A list of types, each encoding the required information to generate an accessor method, grabbing data from the sphere.

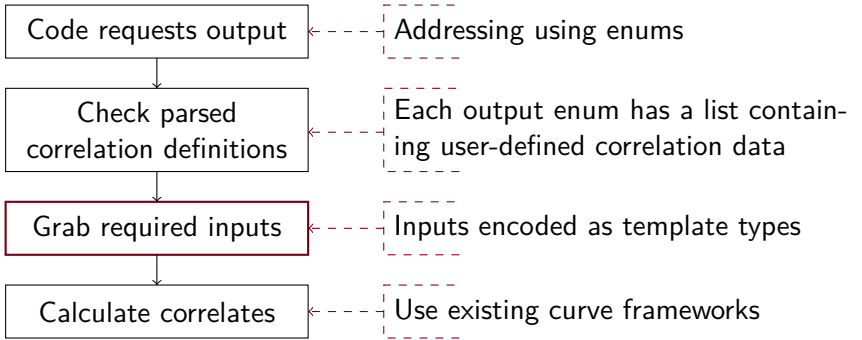
Basic Template Design

The Main Struct


```
1  enum class SphereRenderOutputs { Radius, Velocity, MAX };
2
3  template<typename OutputEnum, typename InputType
4      typename InputGrab...>
5      struct Correlations {
6          using InputGrabTuple = tuple<InputGrab...>;
7          array<ParseData, OutputEnum::MAX> userDefs;
8          float getValue(InputType in, OutputEnum out);
9      };
```

Contains the parsed data for each correlation, stored by output for fast access.

Solution Concept



Basic Template Design

Getting the Input


```
1  template<size_t... idx>
2  float getInput(size_t inputIdx, InputType input,
3      index_sequence<idx...>) {
4      float result = 1.f;
5      ((idx == inputIdx ?
6          (result = tuple_element_t<idx, InputGrabTuple>
7          ::grab(input)), true : false) || ...);
8      return result;
9  }
```

Basic Template Design

Getting the Input


```
1  template<size_t... idx>
2  float getInput(size_t inputIdx, InputType input,
3      index_sequence<idx...>) {
4      float result = 1.f;
5      ((idx == inputIdx ?
6          (result = tuple_element_t<idx, InputGrabTuple>
7          ::grab(input)), true : false) || ...);
8      return result;
9  }
```

`InputType input` is the input object iself (i.e. the sphere object)

Basic Template Design

Getting the Input


```
1  template<size_t... idx>
2  float getInput(size_t inputIdx, InputType input,
3      index_sequence<idx...>) {
4      float result = 1.f;
5      ((idx == inputIdx ?
6          (result = tuple_element_t<idx, InputGrabTuple>
7          ::grab(input)), true : false) || ...);
8      return result;
9  }
```

C++ intricacies require use to use an `index_sequence` to allow allows iteration through the compile-time list of input grabbers.

Basic Template Design

Getting the Input

```
1  template<size_t... idx>
2  float getInput(size_t inputIdx, InputType input,
3      index_sequence<idx...>) {
4      float result = 1.f;
5      ((idx == inputIdx ?
6          (result = tuple_element_t<idx, InputGrabTuple>
7          ::grab(input)), true : false) || ...);
8      return result;
9  }
```

This fold expression generates a switch-case statement at compile-time to access the `grab` method of the `inputIdx`-th input grabber.


```
1 template<size_t... idx>
2 fl
3
4
5
6
7
8
9 }
```

Optimization Notes

Because all input grabbers are specified and processed at compile-time, the entire input-request logic is inlineable, and does not need any runtime lookups of input grabber functions or otherwise slow dynamic dispatch.

Input Grabbers

Design Goal and Usage


```
1  InputGrabber<&Sphere::Radius>
2  ...
3  InputGrabber<&Sphere::Physics, &PhysicsData::Velocity>
```

Input Grabbers

Design Goal and Usage


```
1 InputGrabber<&Sphere::Radius>
2 ...
3 InputGrabber<&Sphere::Physics, &PhysicsData::Velocity>
```

`&Type::Member` is a pointer-to-member, and can be used to access a member of an object of the given type. Since the pointer-to-member is used as a template parameter, it becomes part of the type information and is not runtime data.

Input Grabbers

Design Goal and Usage


```
1  InputGrabber<&Sphere::Radius>
2  ...
3  InputGrabber<&Sphere::Physics, &PhysicsData::Velocity>
```

A chain of pointer-to-members allows access of data that is not a direct member of the input object.


```
1 Input(
2 ...
3 Input(                                    y>
```

Why not just lambdas instead of pointer-to-members?

- Lambdas as template parameters are C++20-only
- Having static functions instead clutters the namespace
- For simple member access, lambdas are much more verbose and require more redundant syntax

Input Grabbers

Implementation


```
1  template<auto... Grabber> struct InputGrabber {
2      template<typename Current, auto Grab, auto... Others>
3      static auto grab_internal(Current input) {
4          if constexpr (sizeof...(Others) == 0){
5              return input.*Grab;
6          } else {
7              return grab_internal<decltype(input.*Grab)>,
8                  Others...>(input.*Grab);
9          }
10     }
11 };
```

Input Grabbers

Implementation


```
1  template<auto... Grabber> struct InputGrabber {
2      template<typename Current, auto Grab, auto... Others>
3      struct grab_internal {
4          static float grab(Current in) {
5              return Grab(in) + grab_internal<Current, Others...>::grab(in);
6          }
7      };
8      template<typename Current, auto Grab>
9      struct grab_internal<Current, Grab> {
10         static float grab(Current in) {
11             return Grab(in);
12         }
13     };
14 };
```

Requires a simple forwarding method

```
1  static float grab(InputType in) {
2      return grab_internal
3          <InputType, Grabber...>(input);
4  }
```

Input Grabbers

Implementation


```
1  template<auto... Grabber> struct InputGrabber {
2      template<typename Current, auto Grab, auto... Others>
3      static auto grab_internal(Current input) {
4          if constexpr (sizeof...(Others) == 0){
5              return input.*Grab;
6          } else {
7              return grab_internal<decltype(input.*Grab)>,
8                  Others...>(input.*Grab);
9          }
10     }
11 };
```

Access the data pointed to by the pointer-to-member if only one pointer-to-member exists (as this `if` is marked `constexpr`, this is checked at compile time).

Input Grabbers

Implementation


```
1  template<auto... Grabber> struct InputGrabber {
2      template<typename Current, auto Grab, auto... Others>
3      static auto grab_internal(Current input) {
4          if constexpr (sizeof...(Others) == 0){
5              return input.*Grab;
6          } else {
7              return grab_internal<decltype(input.*Grab)>,
8                  Others...>(input.*Grab);
9          }
10     }
11 };
```

If more than one pointer-to-member exists, get the data from the first and forward it to the next pointer-to-member recursively.

Input Grabbers

Implementation


```
1  template<auto... Grabber> struct InputGrabber {
2      template<typename Current, auto Grab, auto... Others>
3
4
5
6
7
8
9          }
10     }
11 };
```

Optimization Notes

Because all `if constexpr` branches are evaluated at template-instantiation time, the compiler can and will optimize this into simple submember accesses as if it were hardcoded.


```
1  template<auto ... Grabber> struct InputGrabber {
2
3
4
5
6
7
8
9
10
11 };
```

Further Expansion

The logic shown here was simplified. The actual code, using more `if constexpr` branches is able to evaluate more than just pointer-to-members, including but not limited to:

- pointer-to-member-functions
- array / tuple indexing
- handling of optional / nullable data

Additional Features

In addition to the presented base functionality, the system handles a number of additional features:

- Automatic generation of code to parse the user definitions from provided string names for in- and outputs
- Convenience functions to generate `constexpr` correlation objects
- A system very similar to inheritance in programming in order to reuse inputs for similar correlations without needing to repeat the input definition.

Real-World Example

Definition

Shortened and simplified excerpt of code of the FreeSpace Open game engine

```
1  enum class WeaponOutputs {
2      LASER_LENGTH_MULT,
3      LASER_RADIUS_MULT,
4      LASER_GLOW_MULT,
5      MAX
6  };
7  make_correlation_definition<weapon, WeaponOutputs>(
8    std::array {
9      std::pair {"Laser Length Mult", WeaponOutputs::LASER_LENGTH_MULT},
10     std::pair {"Laser Radius Mult", WeaponOutputs::LASER_RADIUS_MULT},
11     std::pair {"Laser Glow Mult", WeaponOutputs::LASER_GLOW_MULT},
12   },
13   std::pair {"Base Velocity", InputGrabber<&weapon::weapon_max_vel>{}},
14   std::pair {"Max Hitpoints", InputGrabber<&weapon::weapon_info_index,
15     &Weapon_info, &weapon_info::weapon_hitpoints>{}},
16   std::pair {"Parent Radius", InputGrabber<&weapon::objnum, &Objects,
17     &object::parent, &Objects, &object::radius>{}});
```

Real-World Example

Definition

Shortened and simplified excerpt of code of the FreeSpace Open game engine

```
1  enum class WeaponOutputs {
2      LASER_LENGTH_MULT,
3      LASER_RADIUS_MULT,
4
5
6  };
7  ma
8    s
9
10
11
12   }
13   s                                                    }},
14   s                                                    ,
15     &Weapon_info, &weapon_info::weapon_hitpoints>{}},
16   std::pair {"Parent Radius", InputGrabber<&weapon::objnum, &Objects,
17     &object::parent, &Objects, &object::radius>{}});
```

Results

- Low complexity and lines-of-code to create correlations.
- No redundant handling or parsing code is required.
- Required code is almost entirely descriptive.
- Access code is generated at compile-time with no dynamic runtime dispatch and has thus a very low performance cost.

Including indev branches, the FreeSpace Open game engine has the following usage statistics for the correlation system:

- Number of distinct correlation sets: 8
- Of which are inherited sets with shared inputs: 5
- Number of explicitly coded inputs: 22
- Number of inputs through inherited correlation sets: 25
- Number of total inputs: 47
- Number of outputs: 57
- Total lines of code: 215

Real-World Example

Usage Statistics

Including indev branches, the FreeSpace Open game engine has the

Links

Correlation system in FreeSpace Open:

Example correlation definition in FreeSpace Open:

Example correlation usage in FreeSpace Open:

Conclusion

System achievements:

- The correlation system only needs low-complexity, short, and descriptive code to be used.
  - → It can save significant headache compared to hard-coded approaches.
- Dispatch and accessor generation occurs fully at compile-time, and is thus fully inlineable.
  - → Less performance concern compared to dynamic lookup and dispatch.
- No changes are required to the data structures that are used as data inputs.
  - → Easy retrofitability into existing C++ code.

Future work:

- Turn the correlation system into a standalone library.
  - Detach the correlation system from other in-engine subsystems (such as parsing and curve systems).

If you are interested in using this system yourself, do not hesitate to ask me at birk@bmagnu.net