Uniform Call Syntax for explicit-object member functions

Document #:	D0000R1
Date:	2023-02-19
Project:	Programming Language C++
Audience:	EWG

Reply-to:	Gašper Ažman <gasper.azman@gmail.com>

Contents

1 Introduction

This paper introduces a unification of hidden friends and explicit-object member functions to allow a limited, but hopefully uncontroversial Uniform Call Syntax for them.

Unlike the previous proposals on this topic, this one avoids pretty much all controversy.

2 Motivation

Why we might want to have UFCS in the language has been covered expensively already, and Barry Revzin classified all approaches and issues in [Revz].

The post very helpfully lists much prior art by many of WG21's esteemed members: Glassborow, Sutter, Stroustrup, Coe, Orr, and Maurer; specifically [N1585], [N4165], [N4174], [N4474], [P0079R0], [P0131R0], [P0251R0], [P0301R0] and [P0301R1].

With regards to the taxonomy proposd in [Revz], this paper is sortish in the CS:FreeFindsMember category, but with CS:MemberFindsFree and CS:ExtensionMethods left as a possible future extensions, as they aren't mutually exclusive.

This paper proposes OR:OneRound, but with ambiguities being impossible (ill-formed) due to the way this is done.

3 Proposal

We propose that marking an explicit-object member function as a `friend` (to parrot inline friend function declarations, specifically hidden friends) would also make it callable via free-function argument-dependent-lookup.

Example:

```
struct S {
  friend int f(this S) {
    return 42;
  }
};

int g() {
  S{}.f();  // OK, returns 42
  f(S{});   // OK, same
  (f)(S{}); // Error; f can only be found by ADL
}

int f(S);   // ill-formed, int f(S) conflicts with S::f(S)
int f(int); // OK
```

That's pretty much it.

4 How is this different from prior art?

4.1 There can be no confusion about which function is preferred

There is only one function in the first place.

The `friend` syntax signals the behavior exactly. The declaration of the member function is *also* injected into the type's "hidden" namespace as a hidden friend after notionally removing the keyword `this` from the argument list.

This is OK, because explicit-object member functions have free-function type, and their bodies behave as if they were free functions, so we're not lying. We're doing exactly what it looks like.

4.2 It's precise

You opt-in to UFCS on a per-declaration basis. This matters because UFCS is primarily about enabling generic code to use a given type, and gives precise control about both the free-function and member-function interfaces of a given class. When both interfaces should provide a given signature, this is the only proposal that lets you *just do that and only that*, without impacting other parts of either overload set.

4.3 It's simple and minimal

It just merges two things we already have - hidden friends, and explicit-object member functions. No need to remember which comes first or how a given function is defined - both syntaxes always dispatch to the *only* implementation.

4.4 It's modular

It does not propose, but does not preclude, future extensions into, well, extension methods. See the Future Extensions chapter.

5 Future Extensions

While the author of this proposal is of a mild opinion that Extension Metods (CS:ExtensionMethods) would not carry their weight in C++, this paper is specifically neutral on this topic and reserves the only plausible syntax for them:

```cpp
// Disclaimer: NOT PROPOSED IN THIS PAPER
struct E {};
int h(this E) { return 42; } // look ma, I'm not a member of E
int main() {
   h(E{}); // ok, obviously, since f is declared outside of E
   E{}.h(); // also OK, `f` found by ADL and specifies where to put `this`.
}
```

There is one caveat - in this case, if `S` declares an `f(this S)`, it would conflict at declaration time, since this proposal already specifies that behavior.

6 Questions for EWG

1) are we OK choosing the *OR:OneRound (+no conflicting declarations)* approach, knowing that it eliminates OR:TwoRoundsPreferAsWritten, OR:TwoRoundsMemberFirst and OR:OneRoundPreferMembers for all UFCS-related features in the language?
2) Do we want a different syntax from `friend` to signal exactly what `friend` does in this context?
3) Do we find UFCS eliminates a significant-enough portion of library boilerplate in the cases where a class needs to provide both interfaces for this feature to be worth the implementation cost?

7 FAQ

7.1 Why are you writing another paper about UFCS?

Because this is a novel direction that might actually fit the language.

7.2 Has this been implemented?

No, but given that it uses a syntax that is ill-formed in C++23, and that it only inserts an alias to the same function that otherwise works exactly like a hidden friend, I *really* don't have implementation concerns. Any compiler that implements [P0847R7] will have zero issues implementing this paper.

7.3 Are you going to bring the extension methods paper too?

No. I don't need them, and injecting functions into the space given to the class designer is wrong unless properly scoped. I don't know how to properly scope it. If you do, the only reasonable syntax is above.

7.4 Can I put `this` not on the first argument?

Not yet. I might bring that paper if this one passes, but separately.

8 References

[N1585] Francis Glassborow. 2004-02-05. Uniform Calling Syntax (Re-opening public interfaces).
https://wg21.link/n1585

[N4165] Herb Sutter. 2014-10-04. Unified Call Syntax.
https://wg21.link/n4165

[N4174] Bjarne Stroustrup. 2014-10-11. Call syntax: x.f(y) vs. f(x,y).
https://wg21.link/n4174

[N4474] Bjarne Stroustrup, Herb Sutter. 2015-04-12. Unified Call Syntax: x.f(y) and f(x,y).
https://wg21.link/n4474

[P0079R0] Roger Orr. 2015-09-28. Extension methods in C++.
https://wg21.link/p0079r0

[P0131R0] Bjarne Stroustrup. 2015-09-27. Unified call syntax concerns.
https://wg21.link/p0131r0

[P0251R0] Bjarne Stroustrup, Herb Sutter. 2016-02-11. Unified Call Syntax Wording.
https://wg21.link/p0251r0

[P0301R0] Jens Maurer. 2016-03-04. Wording for Unified Call Syntax.
https://wg21.link/p0301r0

[P0301R1] Jens Maurer. 2016-03-21. Wording for Unified Call Syntax (revision 1).
https://wg21.link/p0301r1

[P0847R7] Barry Revzin, Gašper Ažman, Sy Brand, Ben Deane. 2021-07-14. Deducing this.
https://wg21.link/p0847r7

[Revz] Barry Revzin. What is unified function call syntax anyway?
https://brevzin.github.io/c++/2019/04/13/ufcs-history/