Data Structure Visualizers in Visual Studio 2010

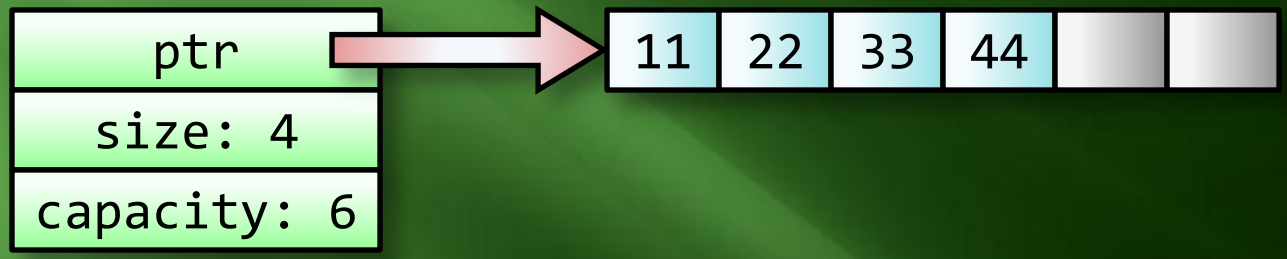
Or: Why "DO NOT MODIFY"

Really Means "HAVE FUN"

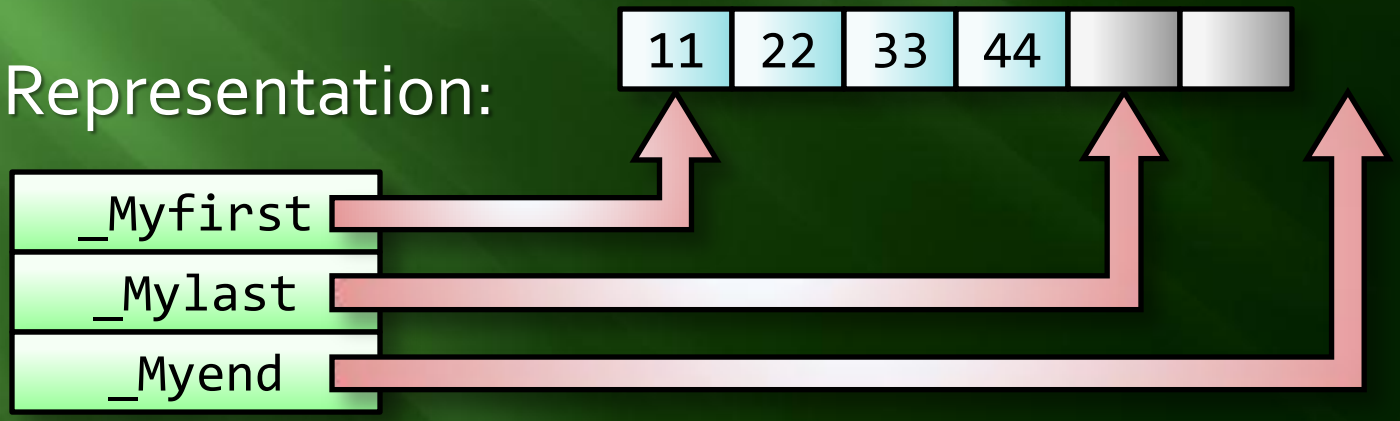
Stephan T. Lavavej
("Steh-fin Lah-wah-wade")
Visual C++ Libraries Developer
stl@microsoft.com

vector: More Than Meets The Eye

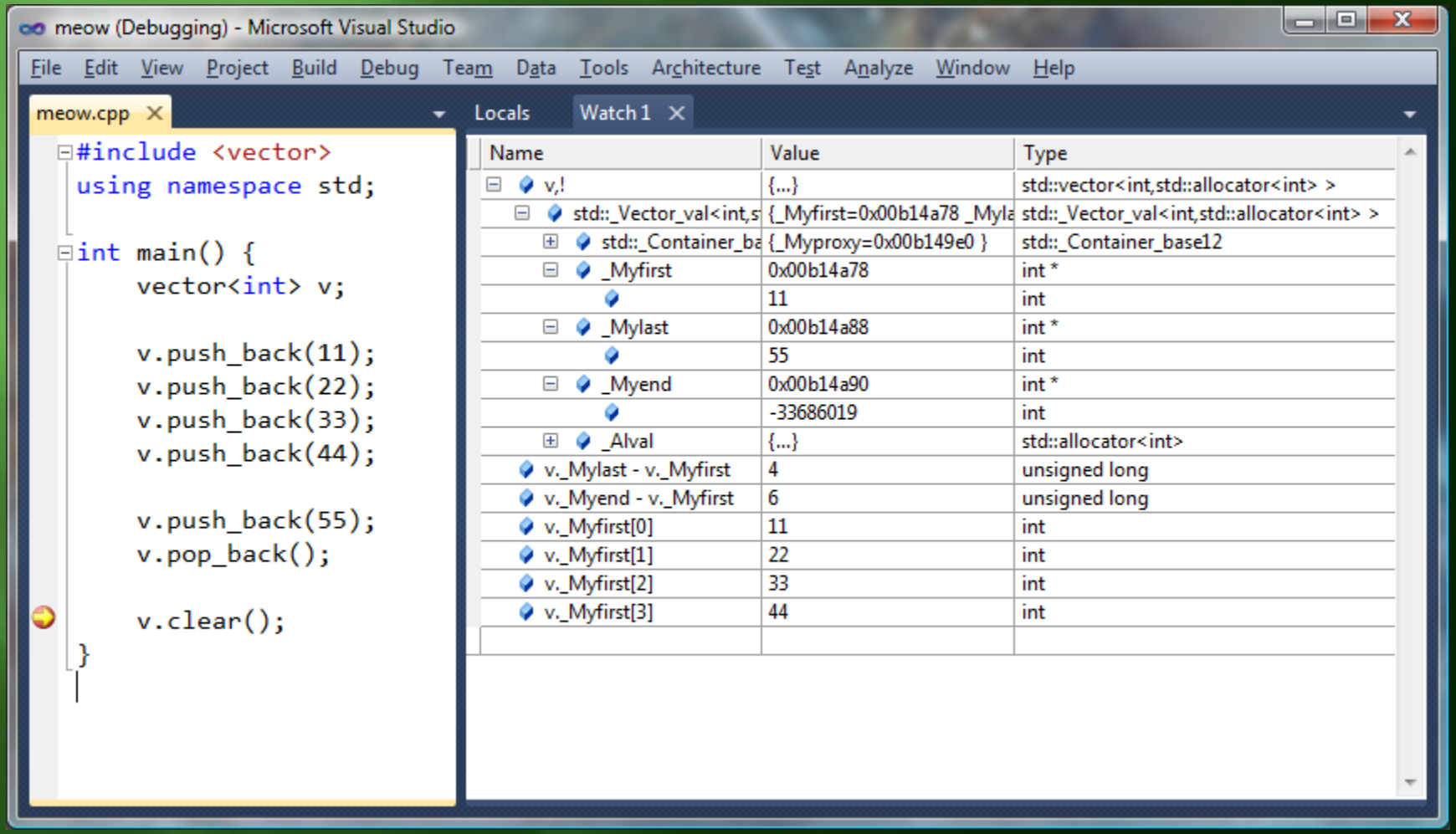
- Appearance:



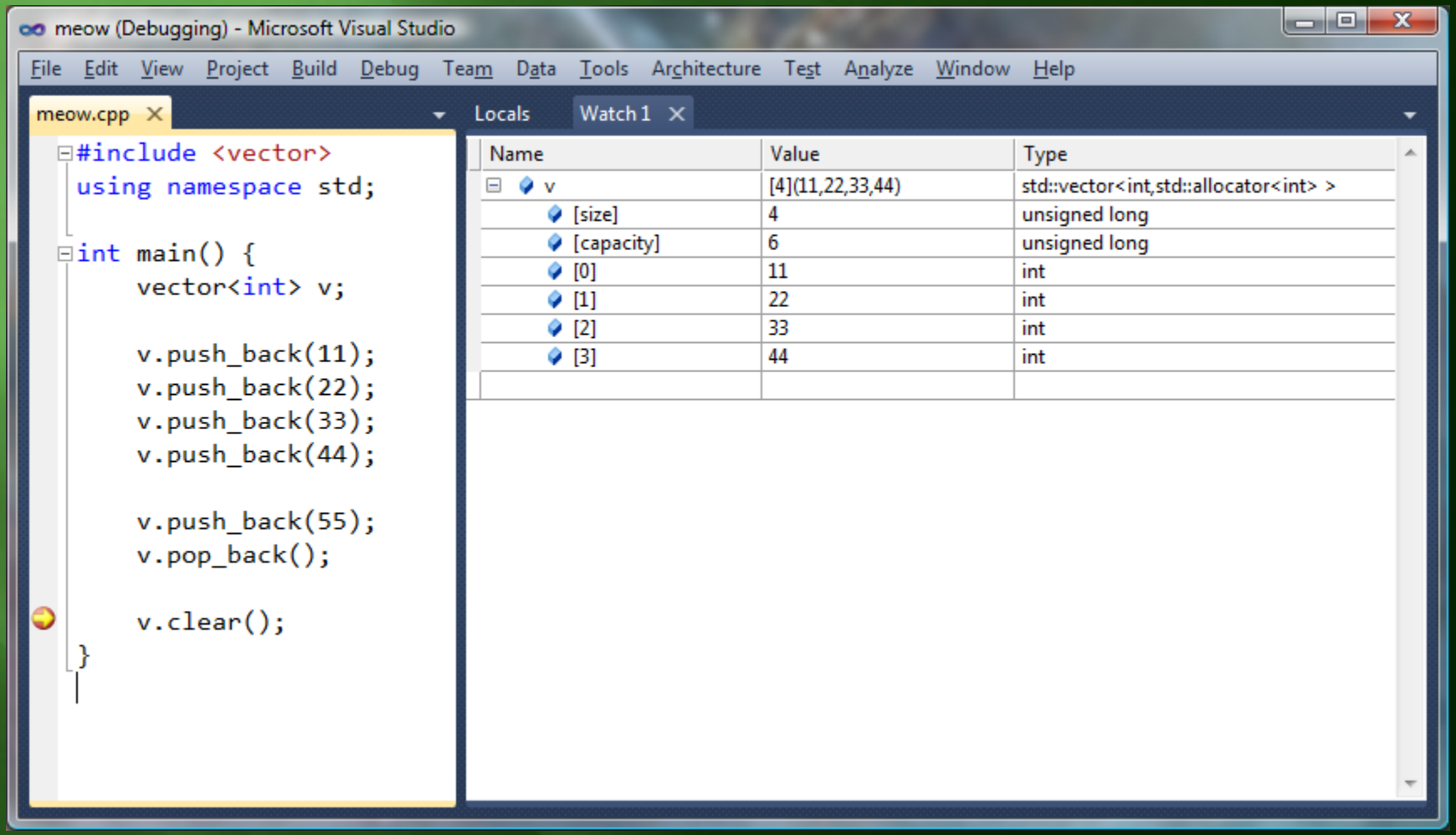
- Representation:



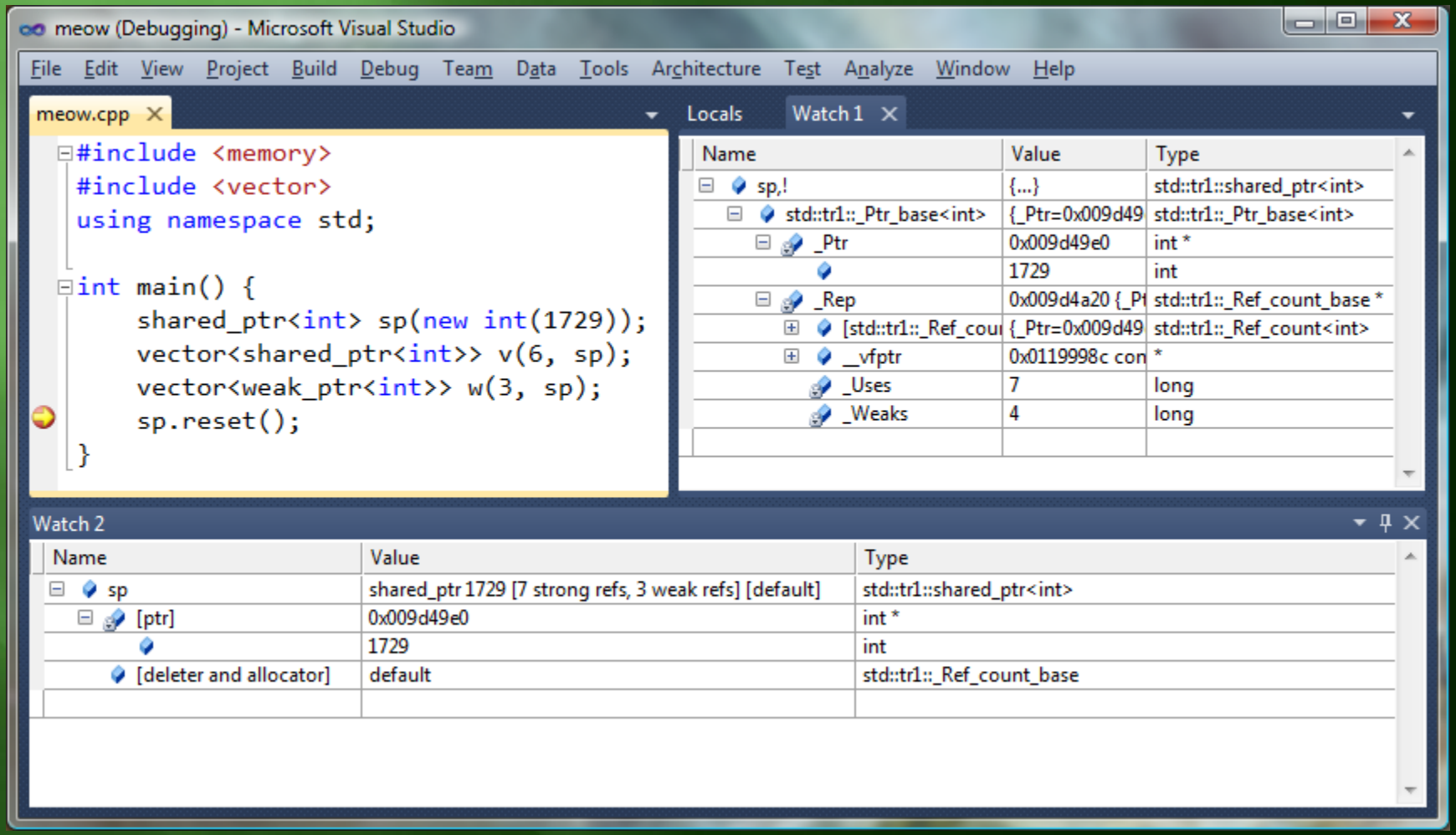
Data Structure Representations Are Ugly



Data Structure Visualizers Are Pretty



Ugly Versus Pretty



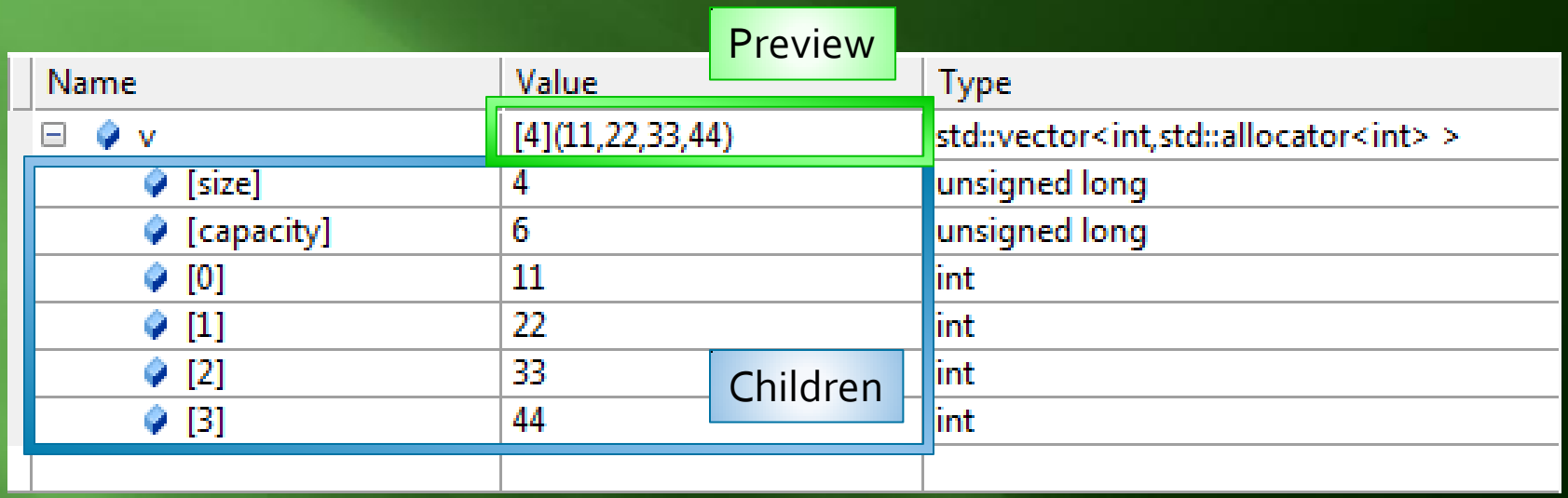
Getting Started

- Visualizers are controlled by:

  ```
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\
  Common7\Packages\Debugger\autoexp.dat
  ```

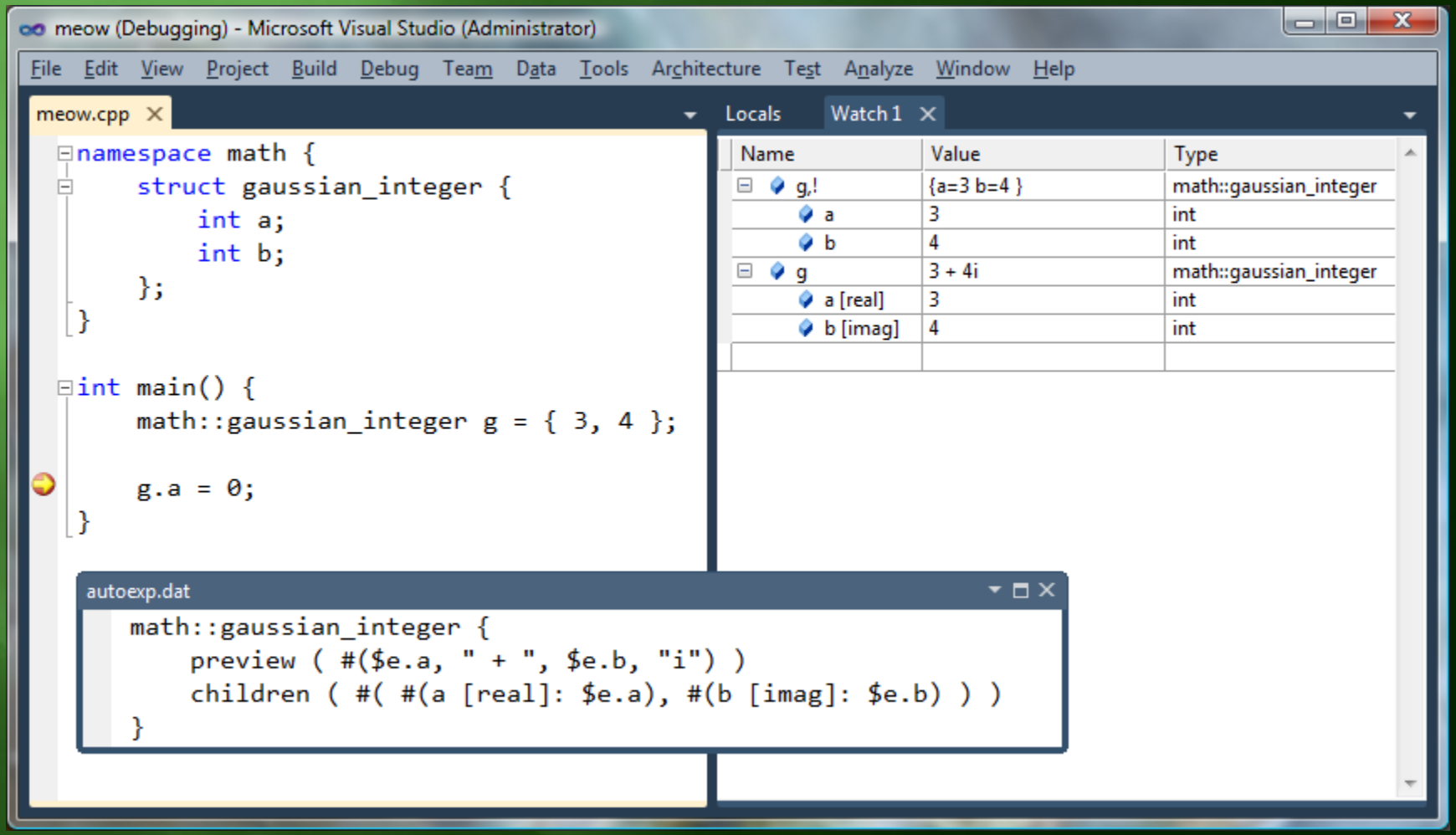
- Undocumented and unsupported
  - You'll find bugs, especially when trying new things
- Traditional Method: Hack this file
  - `; DO NOT MODIFY`
  - Preserve the original file before hacking it!
  - `autoexp.dat` can be edited with VS itself
    - Bypass UAC: Right Click VS > Run As Administrator
- Super Secret Method: Provide another file
  - Put your file's location in the environment variable `_vcee_autoexp`
  - Begin your file with `[Visualizer]`
- Start Debugging (F5) picks up changes
- `var,!` in the Watch Window disables visualization

Previews And Children

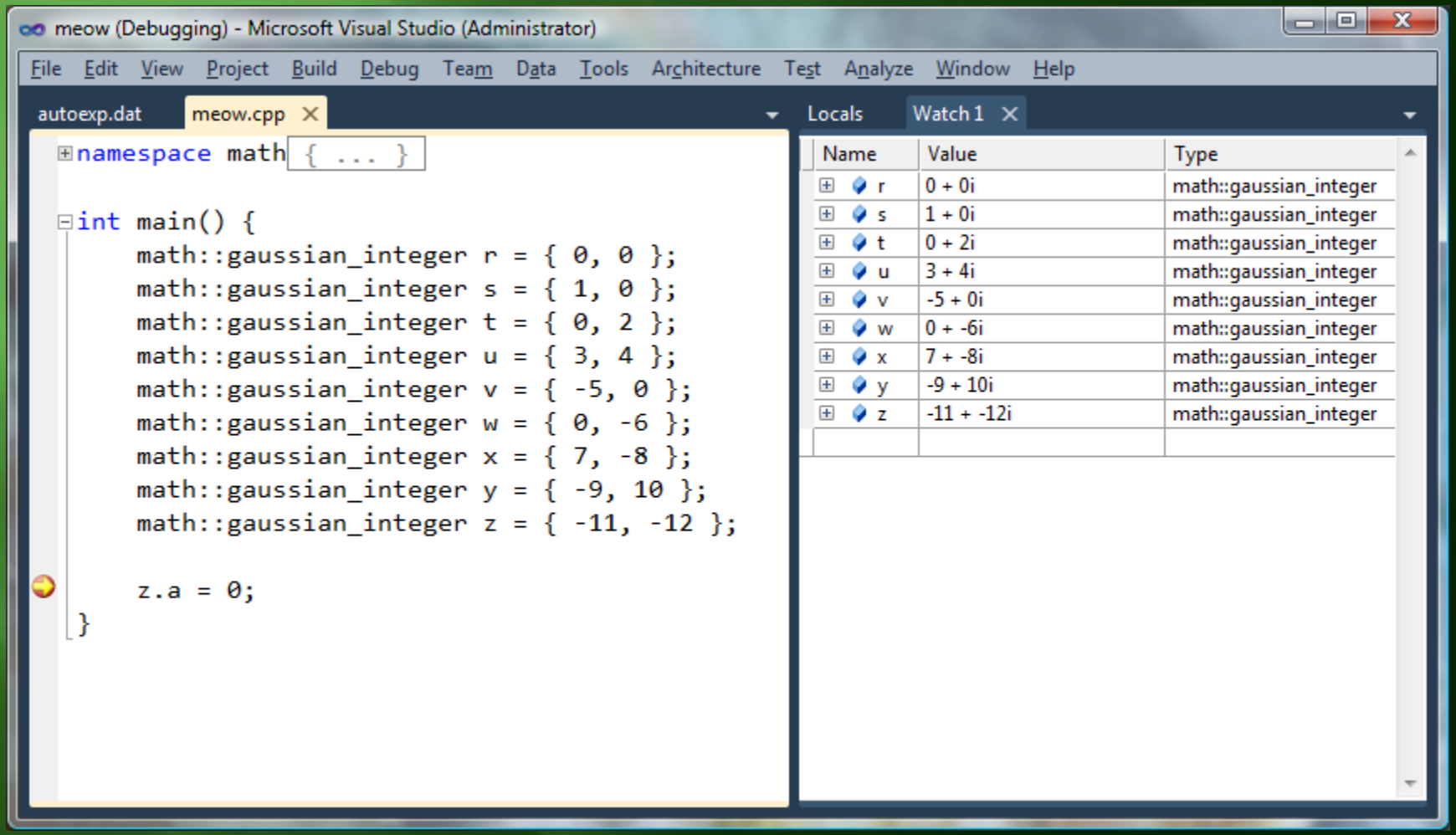


Name	Value	Type
v	[4](11,22,33,44)	std::vector<int,std::allocator<int> >
[size]	4	unsigned long
[capacity]	6	unsigned long
[0]	11	int
[1]	22	int
[2]	33	int
[3]	44	int

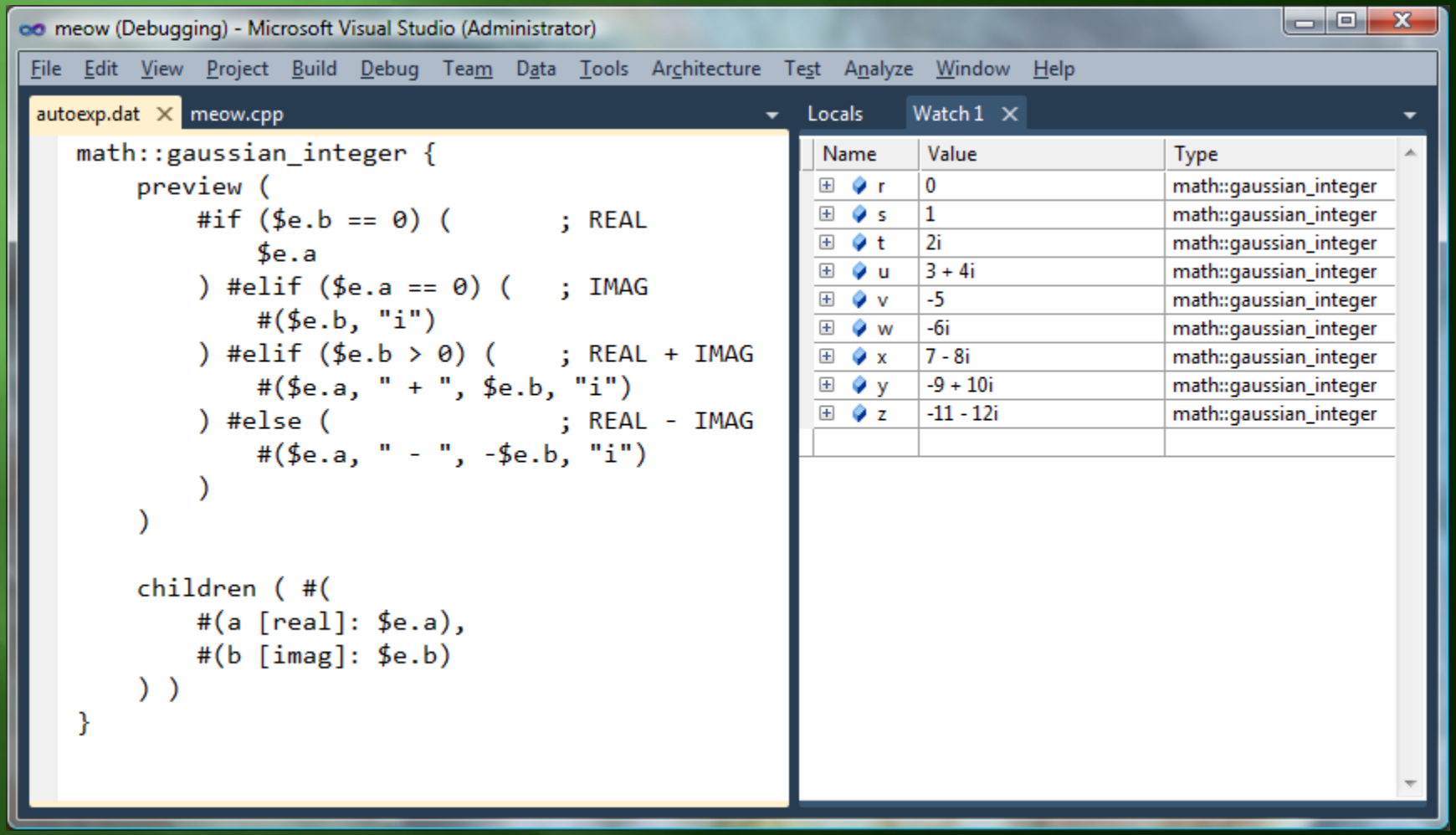
Complex Is Simple (Using preview And children)



Complex Is Complex



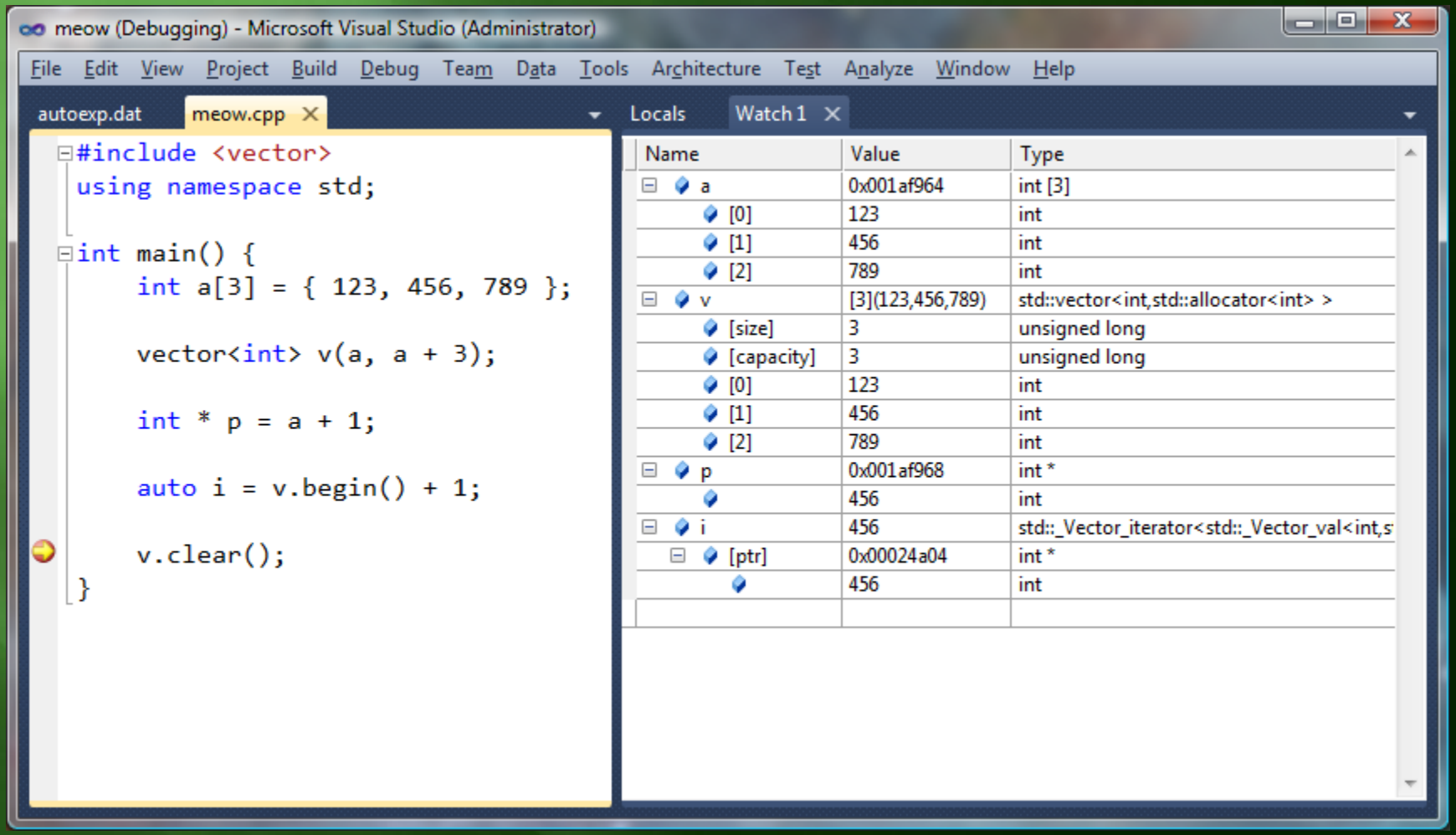
Complex Is Simple Again (Using #if, #elif, And #else)



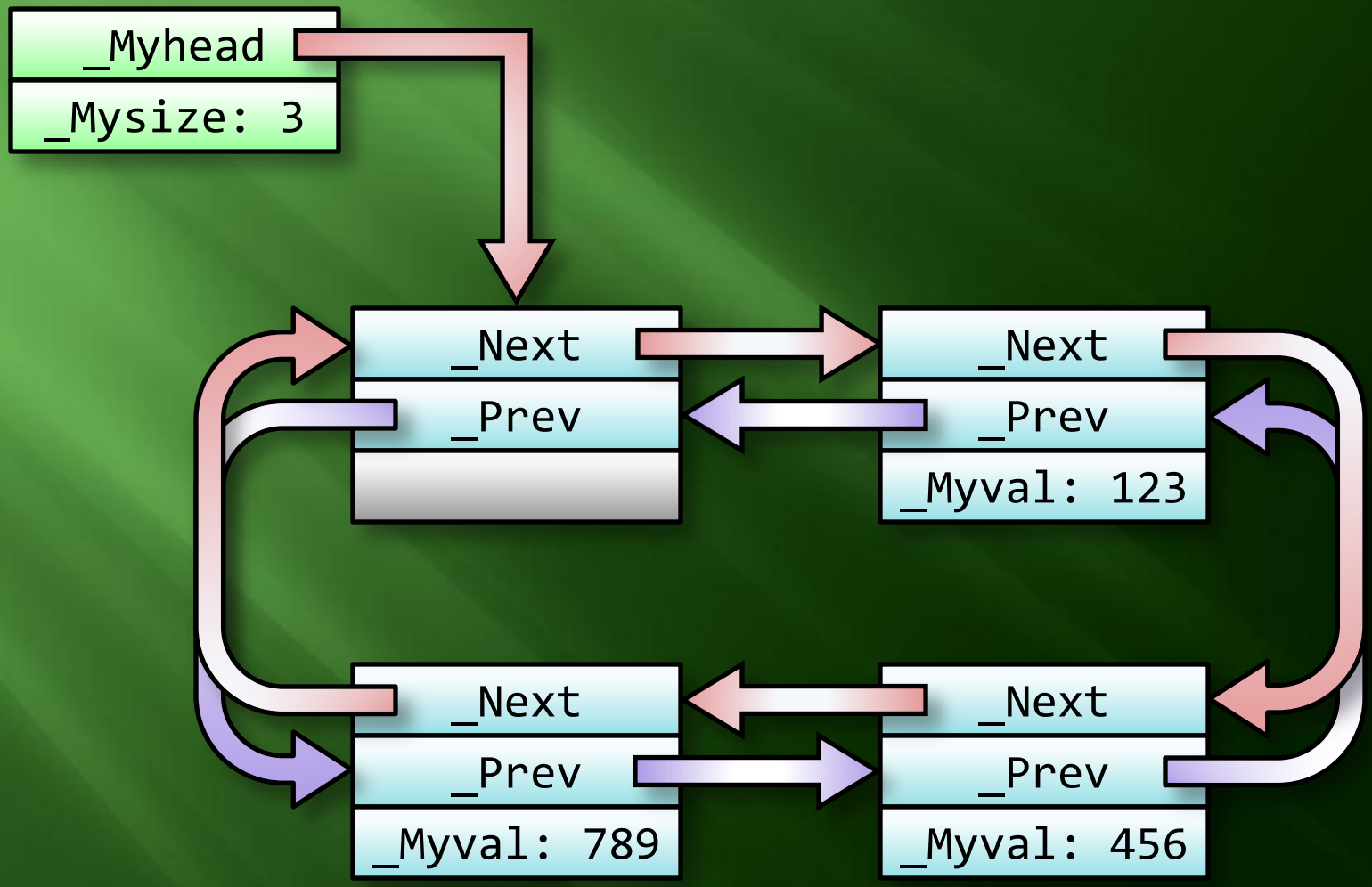
vector's Visualizer (Using #array)

```
std::vector<*>{
    preview ( #(
        "[", $e._Mylast - $e._Myfirst, "](",
        #array(expr: $e._Myfirst[$i],
               size: $e._Mylast - $e._Myfirst), ")"
    ) )
    children ( #(
        #([size] : $e._Mylast - $e._Myfirst),
        #([capacity] : $e._Myend - $e._Myfirst),
        #array(expr: $e._Myfirst[$i],
               size: $e._Mylast - $e._Myfirst)
    ) )
}
std::_Vector_iterator<*>|std::_Vector_const_iterator<*>{
    preview ( *$e._Ptr )
    children ( #([ptr] : $e._Ptr) )
}
```

vector Visualized



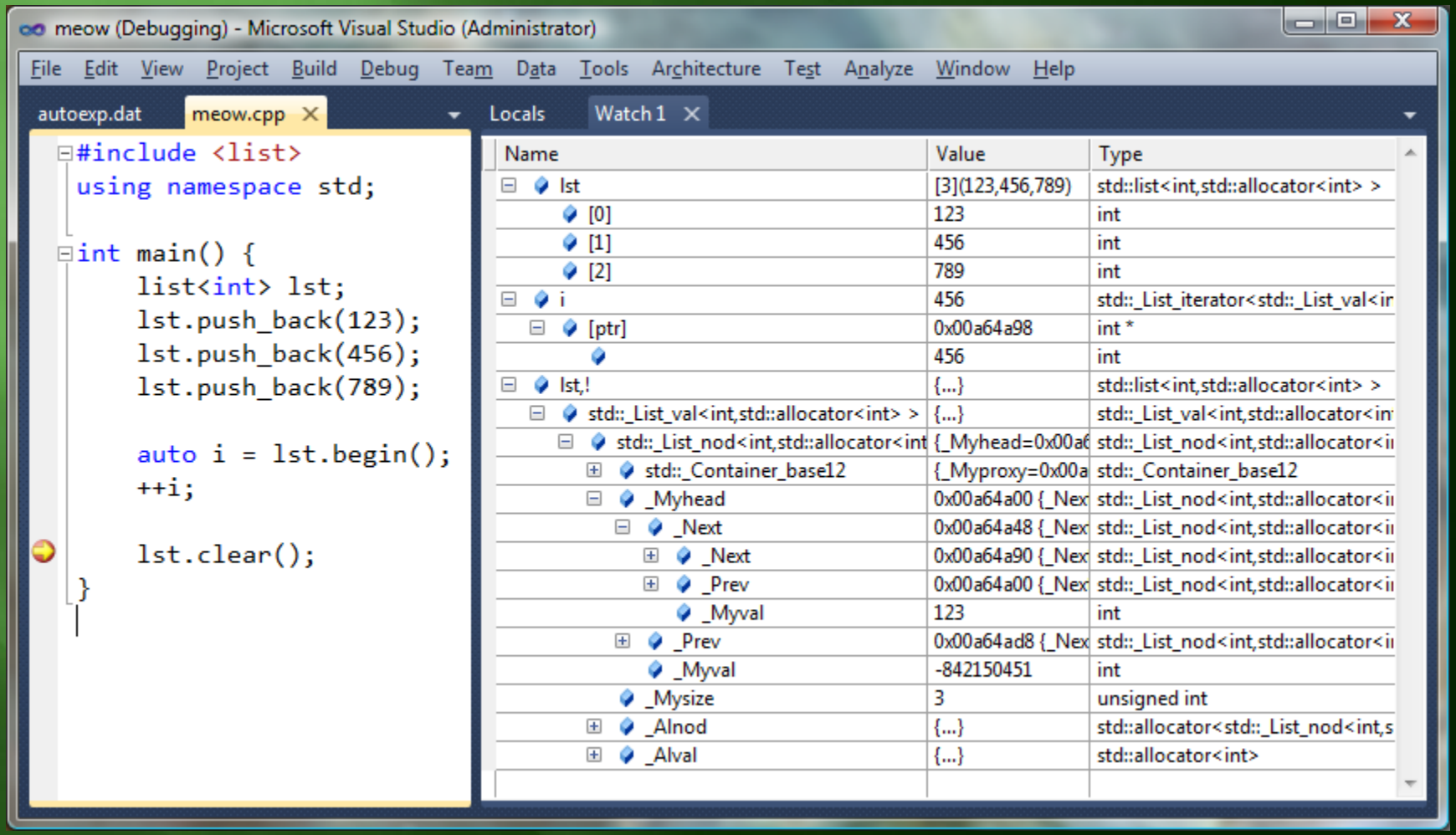
list's Representation



list's Visualizer (Using #list)

```
std::list<*>{
    preview ( #(
        "[", $e._Mysize, "](",
        #list(head: $e._Myhead->_Next,
              size: $e._Mysize,
              next: _Next) : $e._Myval, ")"
    ) )
    children (
        #list(head: $e._Myhead->_Next,
              size: $e._Mysize,
              next: _Next) : $e._Myval
    )
}
std::_List_iterator<*>|std::_List_const_iterator<*>{
    preview ( $e._Ptr->_Myval )
    children ( #([ptr] : &$e._Ptr->_Myval) )
}
```

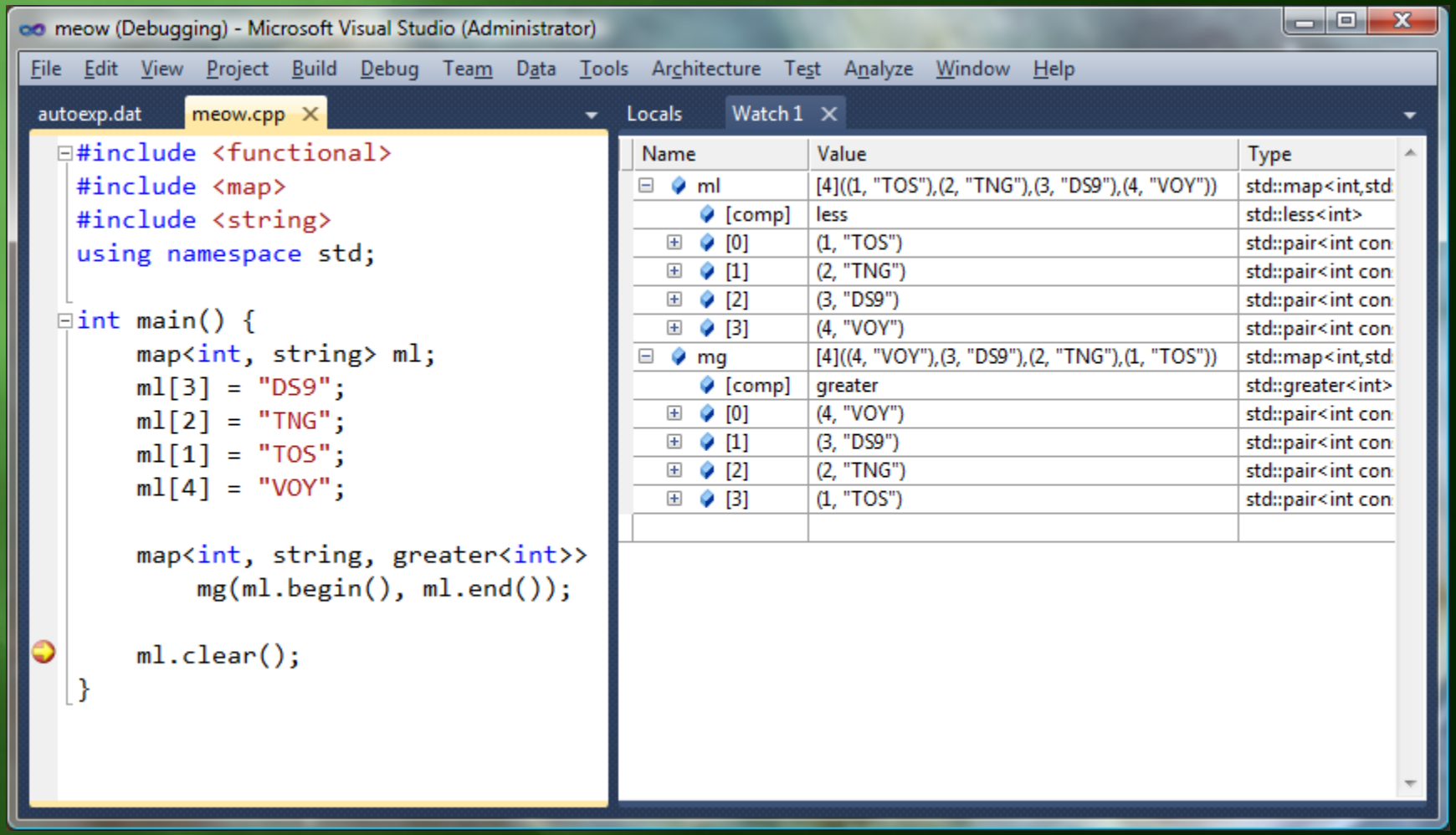
list Visualized



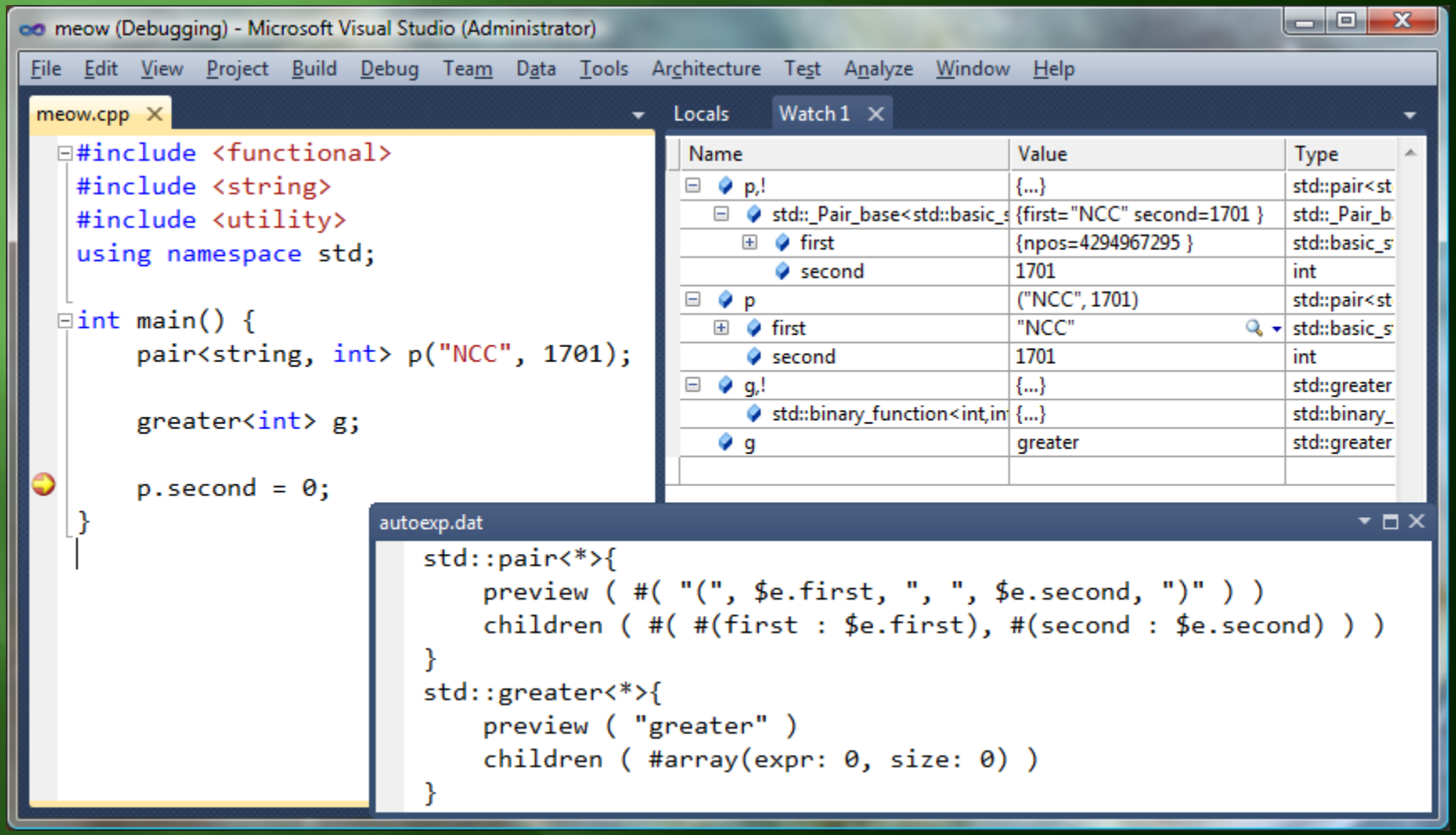
map's Visualizer (Using #tree And A New "Feature")

```
std::map<*>|std::multimap<*>|std::set<*>|std::multiset<*>{
    preview ( #( "[", $e._Mysize, "](",
        #tree(head: $e._Myhead->_Parent, skip: $e._Myhead,
              left: _Left, right: _Right, size: $e._Mysize
        ) : $e._Myval, ")"
    ) )
    children ( #(
        #([comp] : $e.comp),
        #tree(head: $e._Myhead->_Parent, skip: $e._Myhead,
              left: _Left, right: _Right, size: $e._Mysize
        ) : $e._Myval
    ) )
}
std::_Tree_iterator<*>|std::_Tree_const_iterator<*>{
    preview ( $e._Ptr->_Myval )
    children ( #([ptr] : &$e._Ptr->_Myval) )
}
```

map Visualized



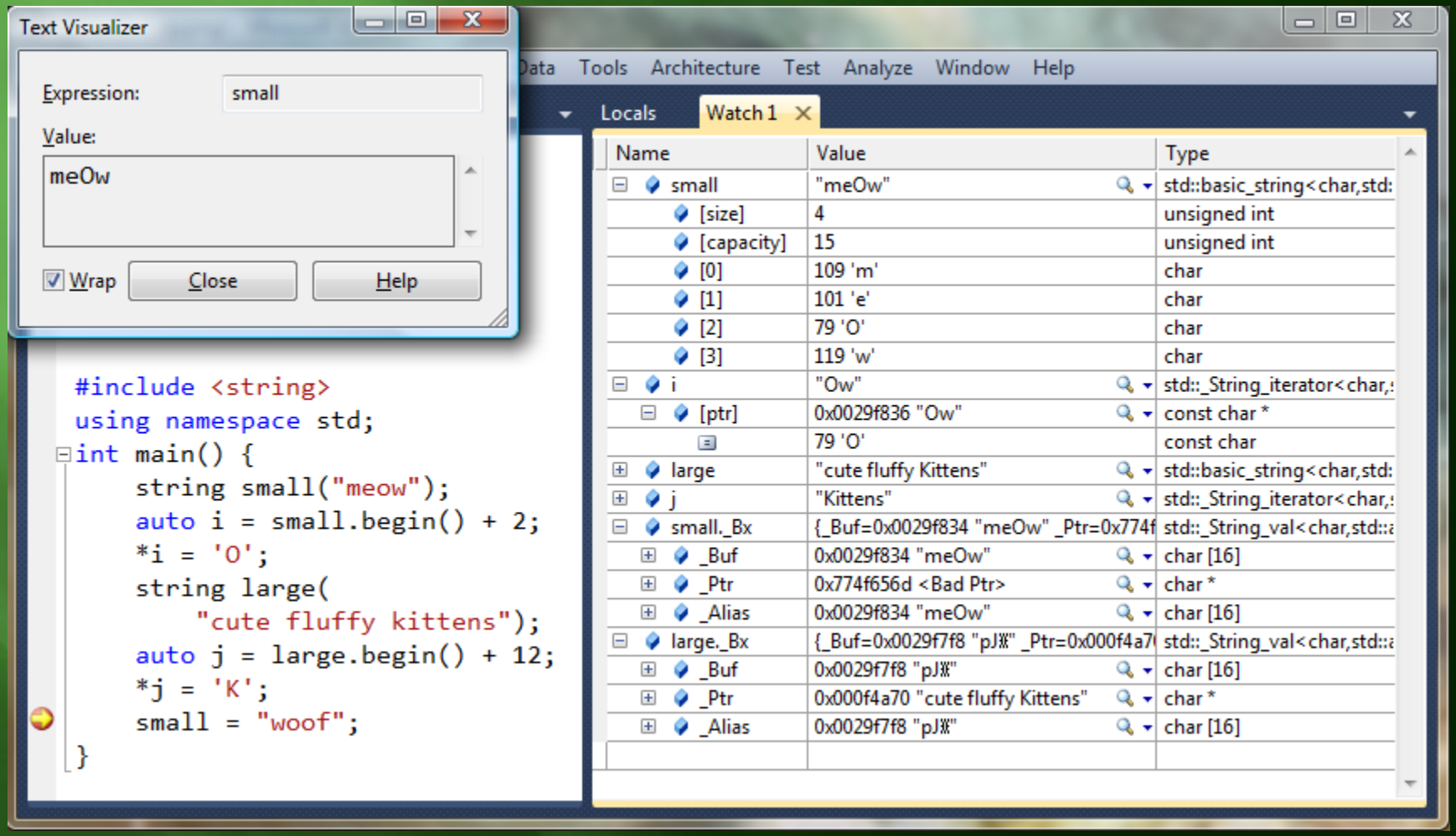
pair and greater's Visualizers (Hiding Base Classes)



string's Visualizer (Using stringview)

```
std::basic_string<char,*>{
  preview    ( #if (($e._Myres) < ($e._BUF_SIZE)) (
    [$e._Bx._Buf,s] ) #else ( [$e._Bx._Ptr,s] ))
  stringview ( #if (($e._Myres) < ($e._BUF_SIZE)) (
    [$e._Bx._Buf,sb] ) #else ( [$e._Bx._Ptr,sb] ))
  children ( #( #([size] : $e._Mysize),
                #([capacity] : $e._Myres),
    #if (($e._Myres) < ($e._BUF_SIZE)) (
      #array(expr: $e._Bx._Buf[$i], size: $e._Mysize)
    ) #else (
      #array(expr: $e._Bx._Ptr[$i], size: $e._Mysize) ) ) ) }
std::_String_iterator<char,*>|std::_String_const_iterator<char,*>{
  preview    ( [$e._Ptr,s] )
  stringview ( [$e._Ptr,sb] )
  children   ( #([ptr] : $e._Ptr) ) }
```

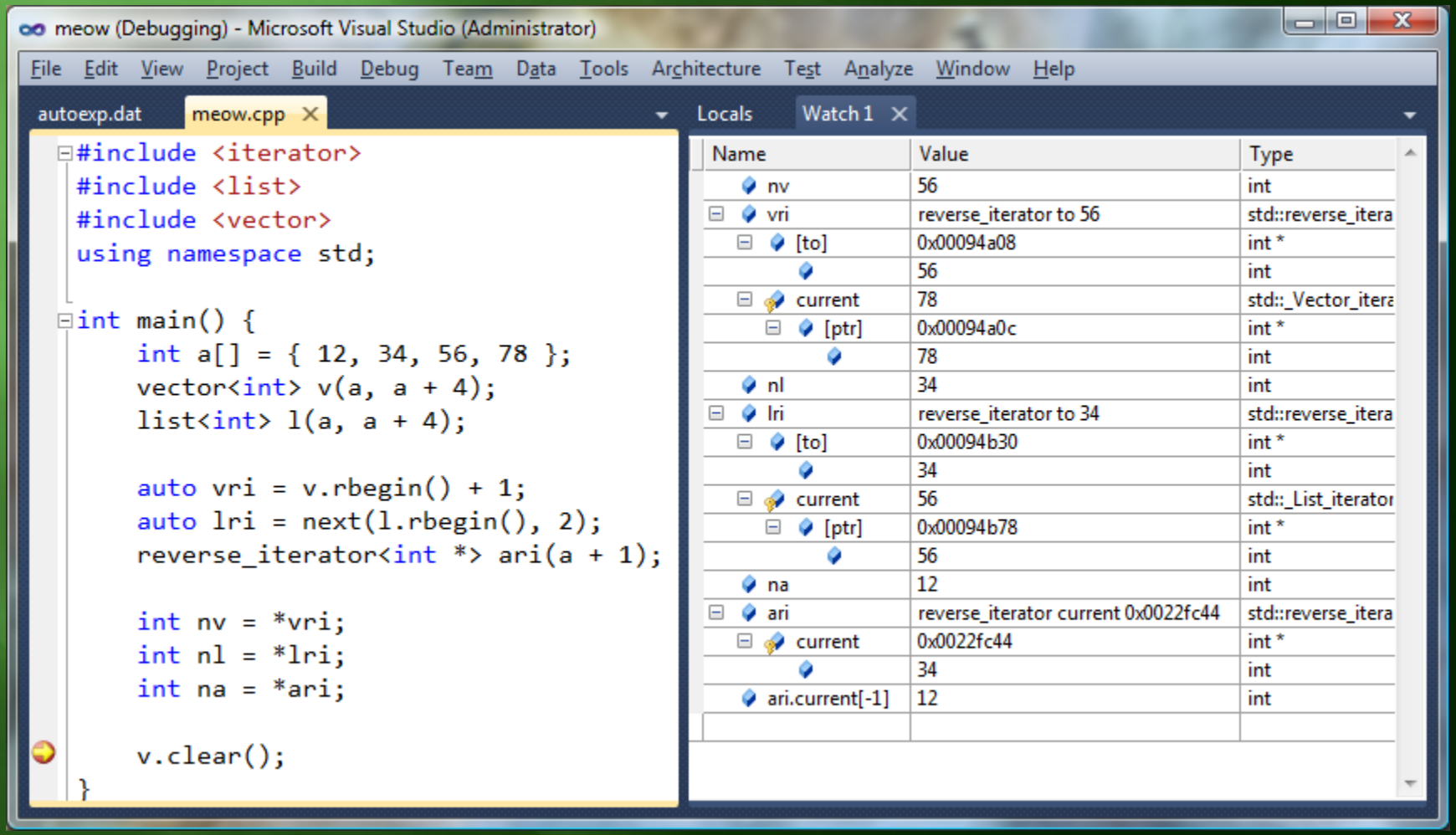
string Visualized



reverse_iterator's Visualizers (Matching Specializations)

```
std::reverse_iterator<std::_Vector_iterator<*> >
|std::reverse_iterator<std::_Vector_const_iterator<*> >{
    preview ( #("reverse_iterator to ", $e.current._Ptr[-1]) )
    children ( #( #([to] : $e.current._Ptr - 1),
                  #(current : $e.current)
) ) }
std::reverse_iterator<std::_List_iterator<*> >
|std::reverse_iterator<std::_List_const_iterator<*> >{
    preview ( #("reverse_iterator to ",
                $e.current._Ptr->_Prev->_Myval) )
    children ( #( #([to] : &$e.current._Ptr->_Prev->_Myval),
                  #(current : $e.current)
) ) }
std::reverse_iterator<*>{
    preview ( #("reverse_iterator current ", $e.current) )
    children ( #(current : $e.current) )
}
```

reverse_iterator Visualized



array's Definition (Dealing With Template Parameters And Macros)

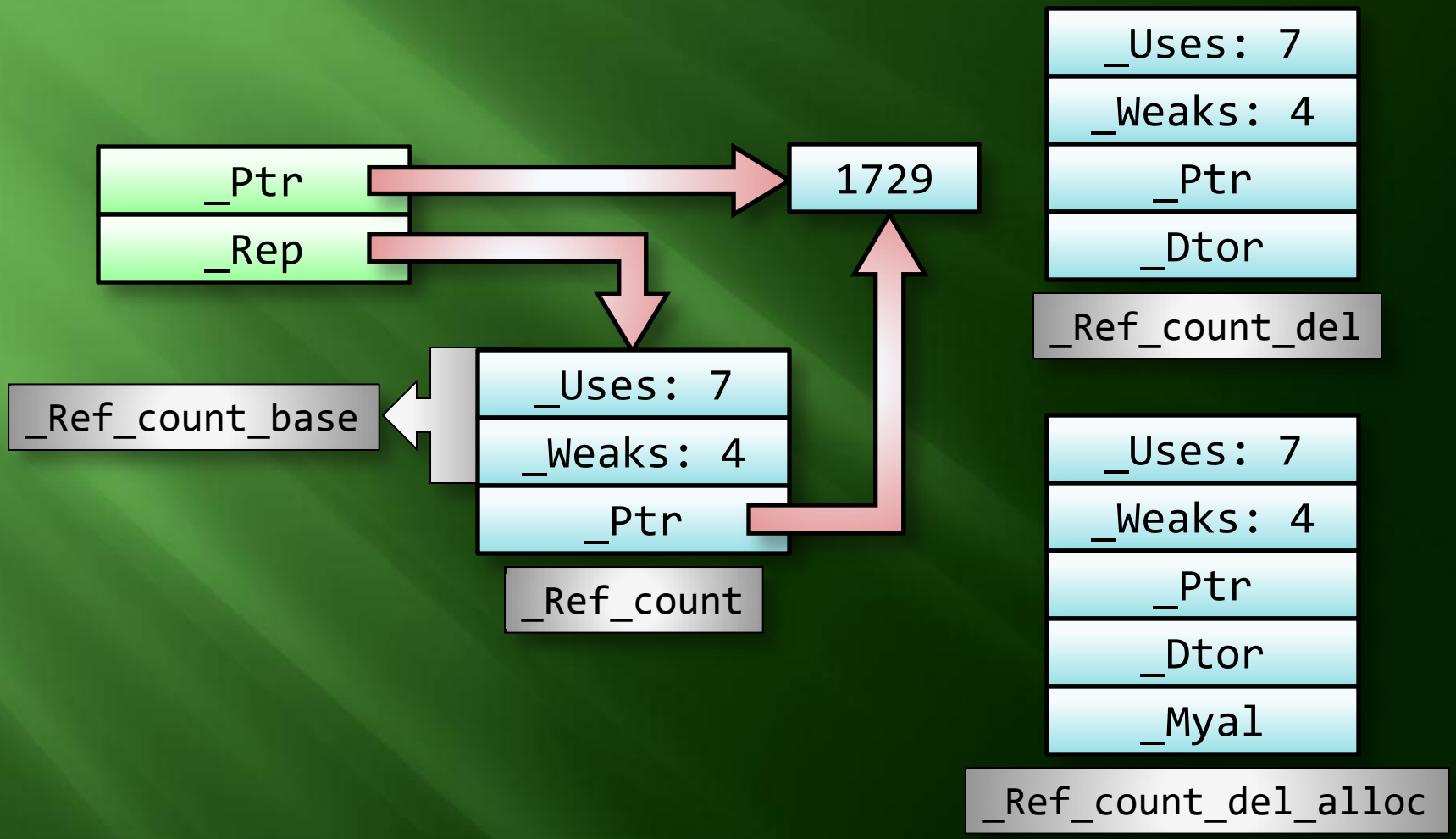
```
template <class _Ty, size_t _Size> class array {
    enum { _EEN_SIZE = _Size };
    _Ty _Elems[_Size];
};

template <class _Ty, size_t _Size>
    class _Array_const_iterator {
    enum { _EEN_SIZE = _Size };
    enum { _EEN_IDL = _ITERATOR_DEBUG_LEVEL };
#if _ITERATOR_DEBUG_LEVEL == 0
    pointer _Ptr;
#else
    pointer _Ptr;
    size_t _Idx;
#endif
};
```

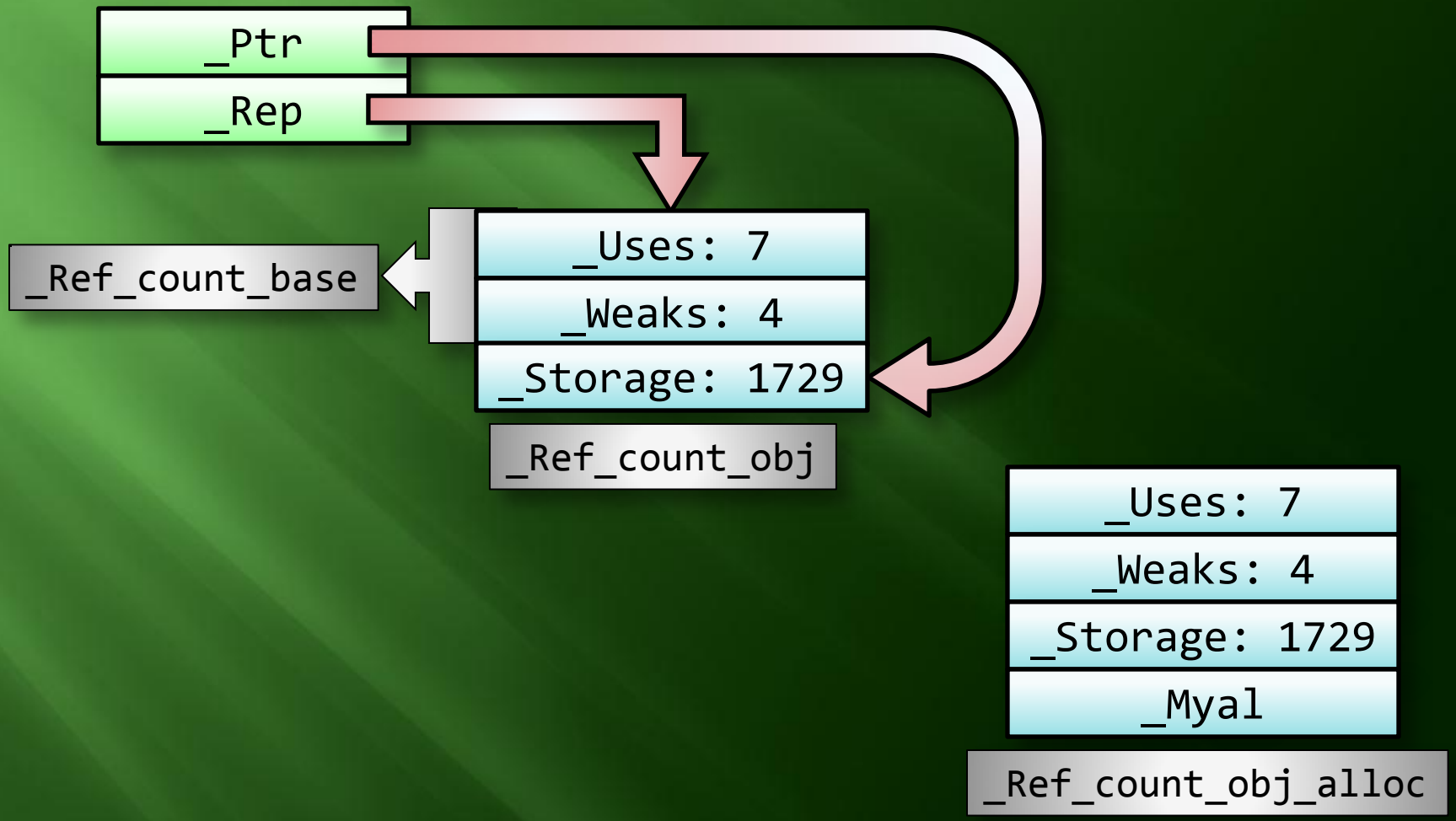
array's Visualizer

```
std::tr1::array<*>{
  preview ( #( "[", $e._EEN_SIZE, "](",
    #array(expr: $e._Elems[$i], size: $e._EEN_SIZE), ")" ) )
  children ( #array(expr: $e._Elems[$i], size: $e._EEN_SIZE) )
}
std::_Array_iterator<*>|std::_Array_const_iterator<*>{
  preview (
    #if ($e._EEN_IDL == 0) ( *$e._Ptr )
    #elif ($e._Idx == $e._EEN_SIZE) ( "end" )
    #else ( $e._Ptr[$e._Idx] ) )
  children (
    #if ($e._EEN_IDL == 0) ( #([ptr] : $e._Ptr) )
    #elif ($e._Idx == $e._EEN_SIZE) (#array(expr: 0, size: 0))
    #else ( #([ptr] : $e._Ptr + $e._Idx) ) )
}
```

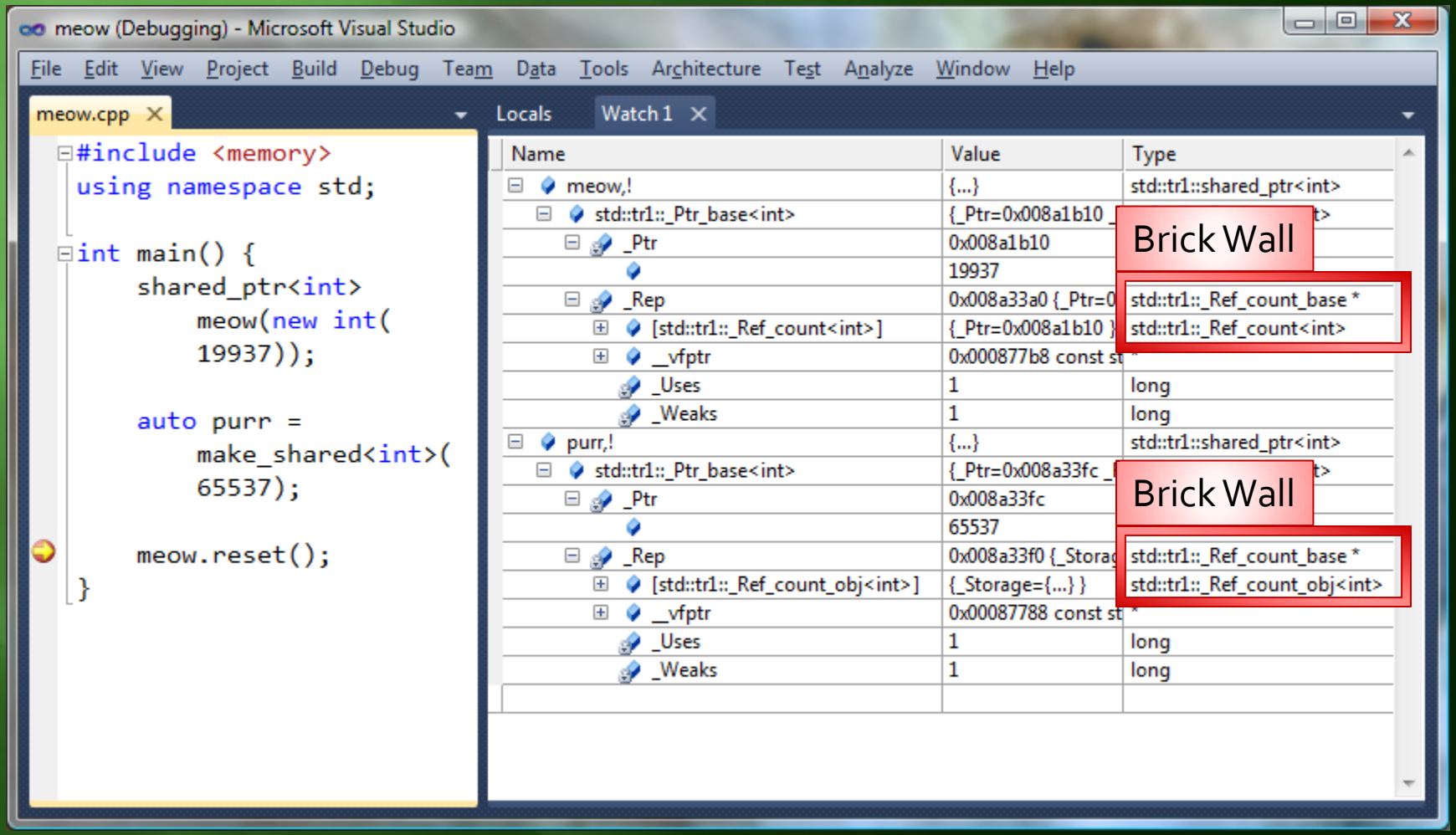
shared_ptr: Refcounts In Disguise



make_shared<T>() And allocate_shared<T>()



The Pointer-To-Base Brick Wall



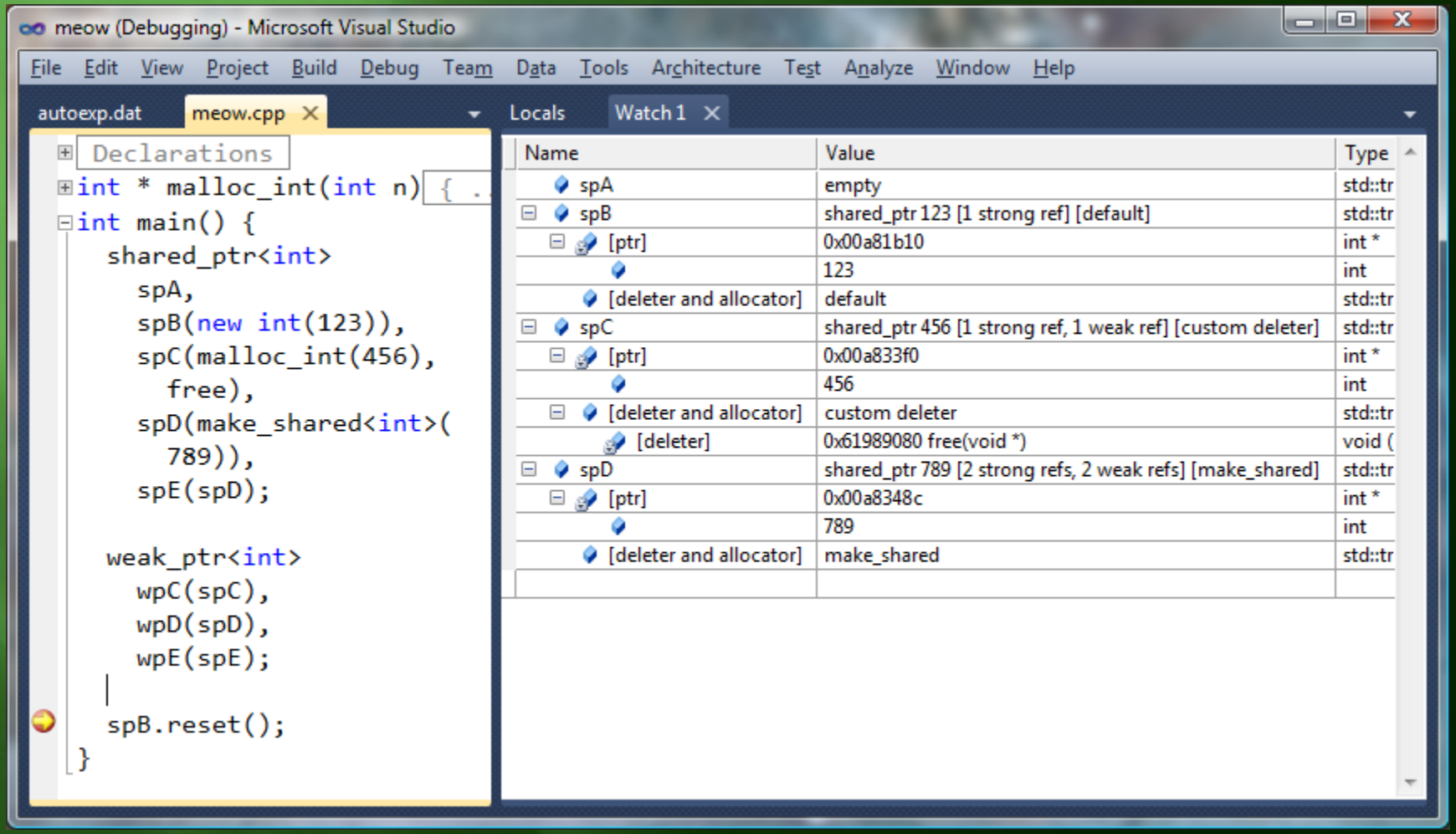
New Feature: Pointer-To-Base X-Ray Vision

```
std::tr1::_Ref_count<*>{
    preview ( "default" )
    children ( #array(expr: 0, size: 0) ) }
std::tr1::_Ref_count_del<*>{
    preview ( "custom deleter" )
    children ( #([deleter] : $e._Dtor) ) }
std::tr1::_Ref_count_del_alloc<*>{
    preview ( "custom deleter, custom allocator" )
    children ( #( #([deleter] : $e._Dtor),
                  #([allocator] : $e._Myal) ) ) }
std::tr1::_Ref_count_obj<*>{
    preview ( "make_shared" )
    children ( #array(expr: 0, size: 0) ) }
std::tr1::_Ref_count_obj_alloc<*>{
    preview ( "allocate_shared" )
    children ( #([allocator] : $e._Myal) ) }
```

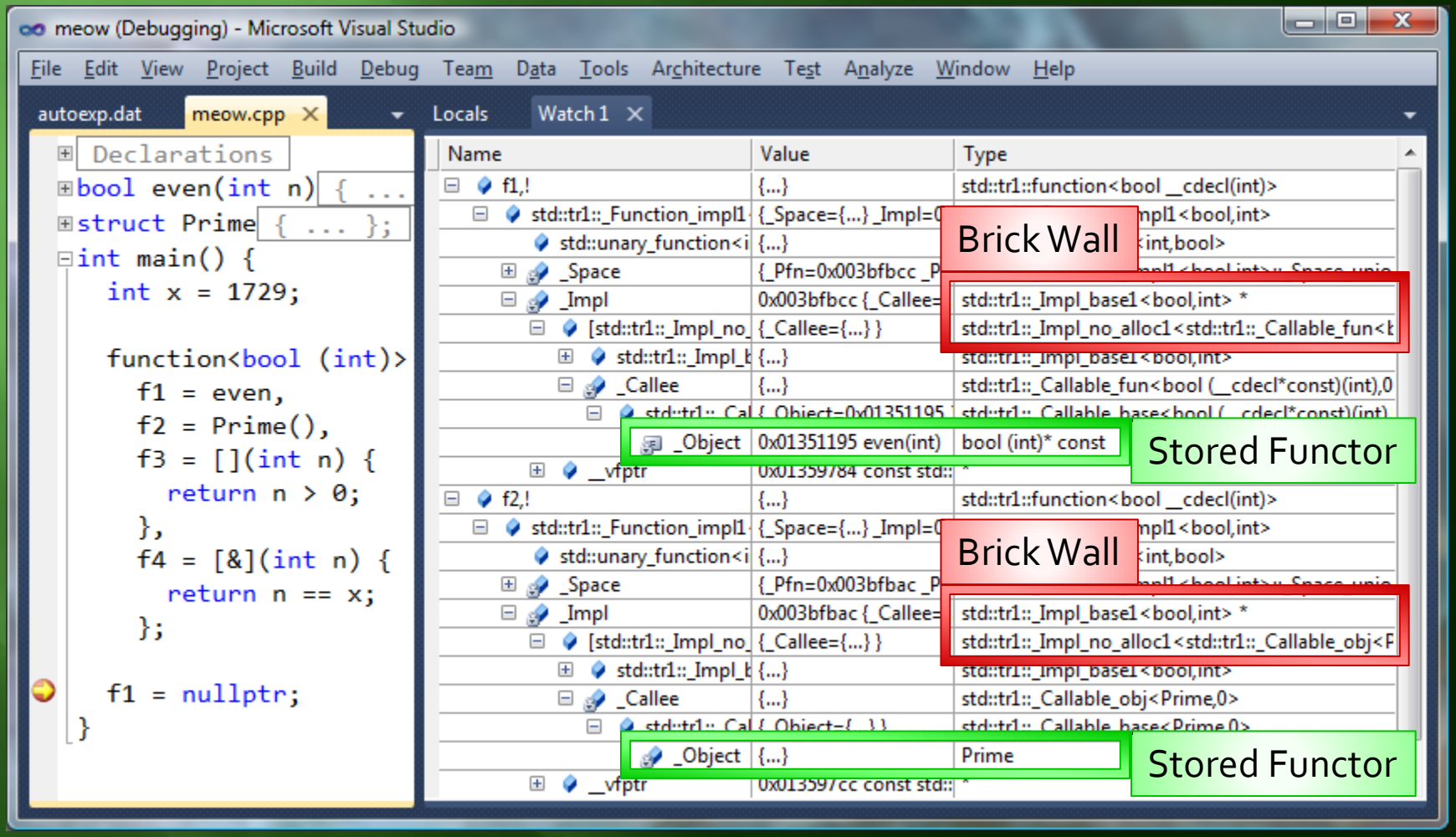
shared_ptr's Visualizer

```
std::tr1::shared_ptr<*>{
    preview ( #if ($e._Ptr == 0) ( "empty" ) #else ( #(
        "shared_ptr ", *$e._Ptr, " [", $e._Rep->_Uses,
        #if ($e._Rep->_Uses == 1) (" strong ref")
                                  #else (" strong refs"),
        #if ($e._Rep->_Weaks - 1 > 0) ( #(
            ", ", $e._Rep->_Weaks - 1,
            #if ($e._Rep->_Weaks - 1 == 1) (" weak ref")
                                           #else (" weak refs")
        ) ),
        "] [", *$e._Rep, "]"
    ) ) )
    children ( #if ($e._Ptr == 0) ( #array(expr: 0, size: 0) )
        #else ( #( #([ptr] : $e._Ptr),
                   #([deleter and allocator] : *$e._Rep)
    ) ) )
}
```

shared_ptr Visualized



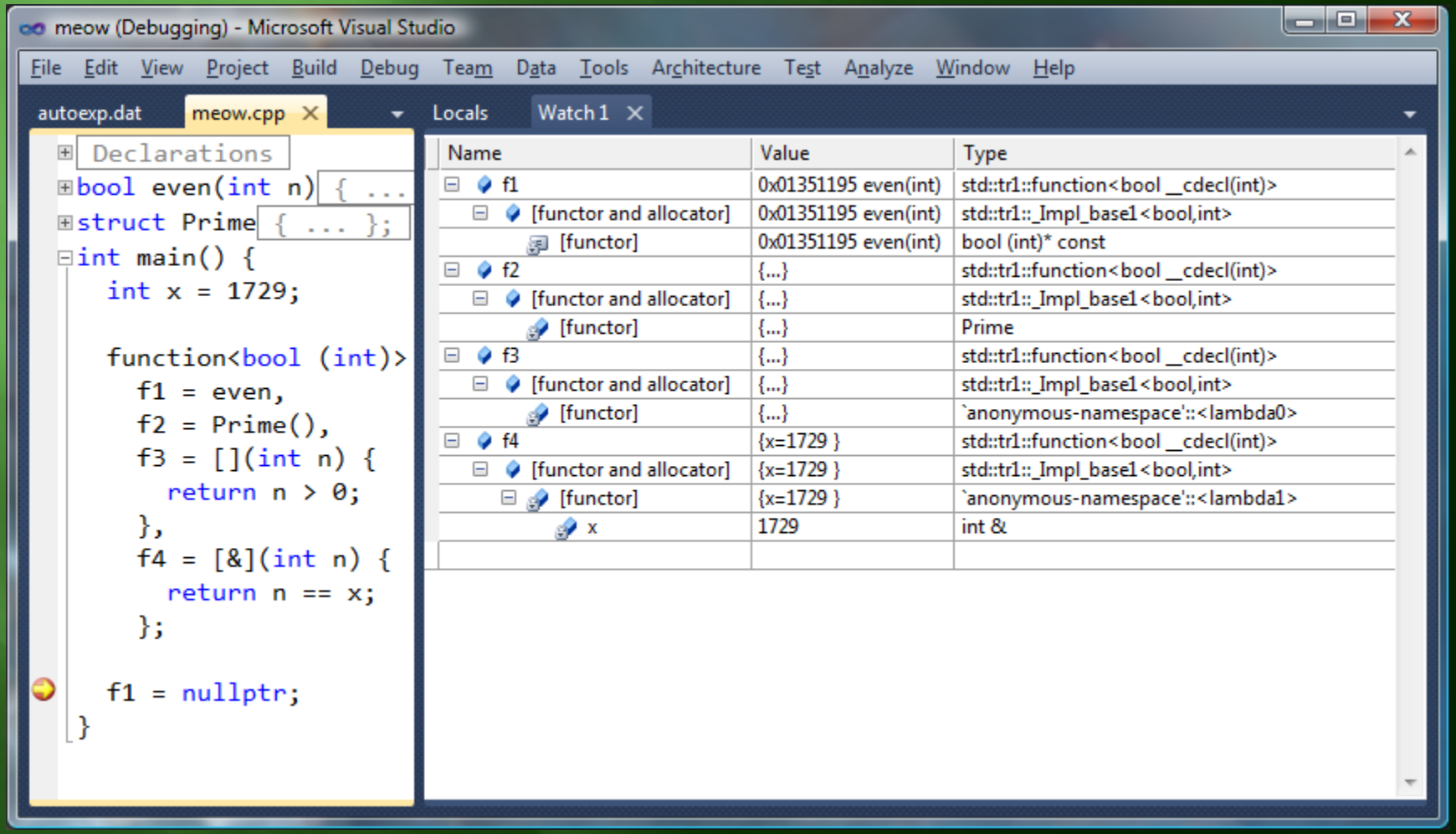
function: Another Brick Wall



function's Visualizer

```
std::tr1::_Impl_no_alloc0<*>|std::tr1::_Impl_no_alloc1<*>|ETC{
    preview ( $e._Callee._Object )
    children ( #([functor] : $e._Callee._Object) )
}
std::tr1::function<*>{
    preview (
        #if ($e._Impl == 0) (
            "empty"
        ) #else (
            *$e._Impl
    ) )
    children (
        #if ($e._Impl == 0) (
            #array(expr: 0, size: 0)
        ) #else (
            #([functor and allocator] : *$e._Impl)
) ) }
```

function Visualized



Questions?

- My E-mail address: stl@microsoft.com
- For more information, read Avery Lee's posts:
  - "Writing custom visualizers for Visual Studio 2005"
    - virtualdub.org/blog/pivot/entry.php?id=120
  - "How to fix debugger visualizers for VS2005 SP1 and VS2008"
    - virtualdub.org/blog/pivot/entry.php?id=172
  - "Watch out for number bases in Visual C++ visualizers"
    - virtualdub.org/blog/pivot/entry.php?id=302