Debugging STL Containers with WinDbg Part 1: Vector

ambrosew 14 Jan 2013 8:06 PM 0

In the prolog of this series, I dabbled a bit on some of the reasons why you'd want to walk the containers in WinDbg yourself instead of using extensions like !stl.

Just to put a stick on the ground for this series, I'm using Visual Studio Ultimate 2012 Version 11.0.50727.1 RTMREL on Windows 8 Version 6.2 Build 9200, and WinDbg 6.2.9200.16384 from Windows Software Development Kit (SDK) for Windows 8. Although I mentioned in the prolog that the containers' types do change a little between Visual C++ releases, the concepts in this series should still apply.

First let's look at vectors. A vector container manages a sequence of elements as a contiguous block of storage, essentially an array that grows on demand. The source in Visual Studio 2012 can be found in C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\crt\src\vector.

Take the following example:

```
C:\Projects\STL>type main.cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    vector<string> ss;
    ss.push_back("1");
    ss.push_back("2");
    ss.push_back("3");
    for (int i = 0; i < (int)ss.size(); i++)
        cout << ss[i] << endl;
}
C:\Projects\STL>cl /EHsc /nologo /W4 /MTd /Zi main.cpp
main.cpp

C:\Projects\STL>main
1
2
3
```

The first step is to look at the size of each element, be it string or other types. In this case it's a string vector:

```
0:000> dt ss
Local var @ 0xd4f938 Type std::vector<std::basic_string<char,std::char_traits<char>,std::allocator<cha
   +0x000 _Myproxy         : 0x00e56418 std::_Container_proxy
   +0x004 _Myfirst         : 0x00e56858 std::basic_string<char,std::char_traits<char>,std::allocator<
   +0x008 _Mylast          : 0x00e568ac std::basic_string<char,std::char_traits<char>,std::allocator<
   +0x00c _Myend           : 0x00e568ac std::basic_string<char,std::char_traits<char>,std::allocator<
```

_Myfirst - pointer to beginning of array.
_Mylast - pointer to current end of sequence.
_Myend - pointer to end of array.

To find out the size of each element, use dt again for the element type, with the -v option to get the size:

```
0:000> dt -v main!std::basic_string<char,std::char_traits<char>,std::allocator<char> >
class std::basic_string<char,std::char_traits<char>,std::allocator<char> >, 166 elements, 0x1c bytes
   +0x000 __BaseClass class std::_String_alloc<0,std::_String_base_types<char,std::allocator<char> > :
   +0x000 __BaseClass class std::_String_val<std::_Simple_types<char> >, 24 elements, 0x1c bytes
   +0x000 __BaseClass struct std::_Container_base12, 9 elements, 0x4 bytes
...
```

There's a whole lot of things that dt spews with -v. Alternatively you can always just use dt and calculate the size yourself:

```
0:000> dt main!std::basic_string<char,std::char_traits<char>,std::allocator<char> >
   +0x000 _Myproxy         : Ptr32 std::_Container_proxy
   +0x004 _Bx              : std::_String_val<std::_Simple_types<char> >::_Bxty
   +0x014 _Mysize          : Uint4B
   +0x018 _Myres           : Uint4B
   =003d31cc npos             : Uint4B
```

At offset 0x18, the last member is Uint4B in size. So the total size is 0x18+4, or 0x1c bytes. Easy enough.

Let's look at the first element. The beginning of the array is at _Myfirst, so just copy the output for _Myfirst from dt ss earlier and do a dt –r on it:

```
0:000> dt ss
Local var @ 0xd4f938 Type std::vector<std::basic_string<char,std::char_traits<char>,std::allocator<cha
   +0x000 _Myproxy         : 0x00e56418 std::_Container_proxy
   +0x004 _Myfirst         : 0x00e56858 std::basic_string<char,std::char_traits<char>,std::allocator<c
   +0x008 _Mylast          : 0x00e568ac std::basic_string<char,std::char_traits<char>,std::allocator<c
   +0x00c _Myend           : 0x00e568ac std::basic_string<char,std::char_traits<char>,std::allocator<c

0:000> dt -r 0x00e56858 std::basic_string<char,std::char_traits<char>,std::allocator<char> >
main!std::basic_string<char,std::char_traits<char>,std::allocator<char> >
   +0x000 _Myproxy         : 0x00e564a8 std::_Container_proxy
      +0x000 _Mycont          : 0x00e56858 std::_Container_base12
         +0x000 _Myproxy         : 0x00e564a8 std::_Container_proxy
      +0x004 _Myfirstiter     : (null)
   +0x004 _Bx              : std::_String_val<std::_Simple_types<char> >::_Bxty
      +0x000 _Buf             : [16]  "1"
      +0x000 _Ptr             : 0xcdcd0031  "--- memory read error at address 0xcdcd0031 ---"
      +0x000 _Alias           : [16]  "1"
   +0x014 _Mysize          : 1
   +0x018 _Myres           : 0xf
   =003d31cc npos             : 0xffffffff
```

For strings, the most interesting member would be _Bx, which is a union type. From C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\crt\src\xstring:

```
union _Bxty
    {    // storage for small buffer or pointer to larger one
    value_type _Buf[_BUF_SIZE];
    pointer _Ptr;
    char _Alias[_BUF_SIZE];    // to permit aliasing
    } _Bx;
```

Your string data will either be in _Buf array for short string, or in the heap pointed to by _Ptr. _Mysize is the current length of the string, and _Myres is the current storage reserved for the string. _Alias is there since Visual Studio 2010 for aliasing.

Let’s go back to vectors.

To find the next element from the first, all we need to do is add the element size to the first element’s address. Here’s the second element in the vector:

```
0:000> dt -r 0x00e56858+0x1c std::basic_string<char,std::char_traits<char>,std::allocator<char> >
main!std::basic_string<char,std::char_traits<char>,std::allocator<char> >
   +0x000 _Myproxy         : 0x00e568e8 std::_Container_proxy
      +0x000 _Mycont          : 0x00e56874 std::_Container_base12
         +0x000 _Myproxy         : 0x00e568e8 std::_Container_proxy
      +0x004 _Myfirstiter     : (null)
   +0x004 _Bx              : std::_String_val<std::_Simple_types<char> >::_Bxty
      +0x000 _Buf             : [16]  "2"
      +0x000 _Ptr             : 0xcdcd0032  "--- memory read error at address 0xcdcd0032 ---"
      +0x000 _Alias           : [16]  "2"
   +0x014 _Mysize          : 1
   +0x018 _Myres           : 0xf
   =003d31cc npos             : 0xffffffff
```

And the third:

```
0:000> dt -r 0x00e56858+0x1c+0x1c std::basic_string<char,std::char_traits<char>,std::allocator<char>
main!std::basic_string<char,std::char_traits<char>,std::allocator<char> >
   +0x000 _Myproxy         : 0x00e56750 std::_Container_proxy
      +0x000 _Mycont          : 0x00e56890 std::_Container_base12
         +0x000 _Myproxy         : 0x00e56750 std::_Container_proxy
      +0x004 _Myfirstiter     : (null)
   +0x004 _Bx              : std::_String_val<std::_Simple_types<char> >::_Bxty
      +0x000 _Buf             : [16]  "3"
      +0x000 _Ptr             : 0xcdcd0033  "--- memory read error at address 0xcdcd0033 ---"
      +0x000 _Alias           : [16]  "3"
   +0x014 _Mysize          : 1
   +0x018 _Myres           : 0xf
   =003d31cc npos             : 0xffffffff
```

And so on and so forth.

If you have a large number of elements to examine, you’d probably want the computer to do the enumeration for you. A debugger script will fit the bill. Here’s an example:

1. Store the following content in a text file named, say, vectorstring.txt, and put it in, say, c:\temp.

```
$$ Initialize our pseudo-registers
```

```
r? $t0=${$arg1}._Myfirst
r? $t1=${$arg1}._Mylast

$$ Go through the elements until _Mylast is reached.
.while (@$t0 != @$t1)
{
  $$ Dump the string content
  da @@c++(&@$t0->_Bx)

  $$ Go to the next element.
  r? $t0=@$t0+1
}
```

2. In the debugger, run the script with argument: $$>a< vectorstring.txt <vector>

```
0:000> $$>a< c:\temp\vectorstring.txt ss
00e5685c  "1"
00e56878  "2"
00e56894  "3"
```

This could be useful either to just look at the data of each element, or walk the vector array for suspicious values (e.g. buffer overruns). Customize the script to your heart's content.

This is just about all I needed in order to peel the vector container. I hope this is a good start of the series for you. As always, happy debugging!