

Developing

EnergyPlus C++

Stuart G. Mentzer
Objexx Engineering, Inc.

C++ Standard Compliance

EnergyPlus & ObjexxFCL are C++11 compliant

C++11 supports cleaner, safer code

ObjexxFCL exploits C++11: `a({3,7}, 9)`

`clang-modernize` to further exploit C++11

C++ Compilers

GCC: C++11 is good / MinGW: Slow deb. link

Clang: C++11 is good but release build chokes

Intel C++: C++11 is OK in 14.0.2: work-arounds

Visual C++ 2013: Many C++/11 bugs

- ObjexxFCL and EnergyPlus work-arounds
- Limited functionality / May be usable for EnergyPlus

IDEs

Eclipse/CDT

Qt Creator

KDevelop

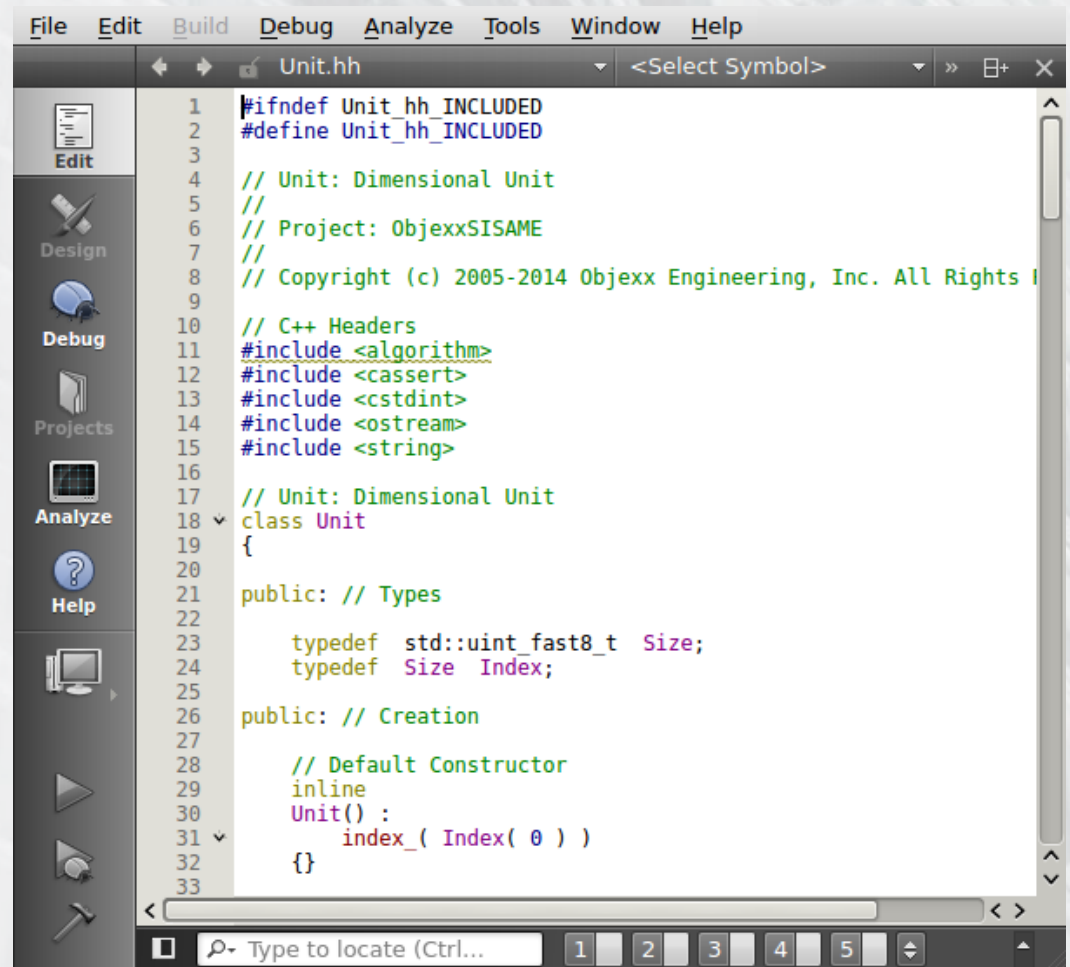
Visual Studio

Code:Blocks

CodeLite

NetBeans

SlickEdit



```
File Edit Build Debug Analyze Tools Window Help
Unit.hh
1 #ifndef Unit_hh_INCLUDED
2 #define Unit_hh_INCLUDED
3
4 // Unit: Dimensional Unit
5 //
6 // Project: ObjexxSISAME
7 //
8 // Copyright (c) 2005-2014 Objexx Engineering, Inc. All Rights Reserved
9
10 // C++ Headers
11 #include <algorithm>
12 #include <cassert>
13 #include <cstdint>
14 #include <ostream>
15 #include <string>
16
17 // Unit: Dimensional Unit
18 class Unit
19 {
20
21 public: // Types
22
23     typedef std::uint_fast8_t Size;
24     typedef Size Index;
25
26 public: // Creation
27
28     // Default Constructor
29     inline
30     Unit() :
31         index_( Index( 0 ) )
32     {}
33
```

Building C++ vs Fortran

C++ is **slower** to compile than Fortran

- Template libs like ObjexxFCL & Boost slow it more
- More/faster CPU cores helps (parallel makes)
- Code reorg could help / Maybe precompiled headers

C++ headers decouple better than Fortran modules => **Faster** builds after impl. changes

C++ debug executables are **much** bigger

Building EnergyPlus

GNU make system provided

- Cross-platform
- Zero-maintenance
 - Builds all sources in dir tree
 - Generates/updates dependencies
 - Cleans obsolete objects
- Collision/parallel safe

Other systems (Cmake, ...) could be added

Building EnergyPlus: Linux

1. Open a console in the repository root
 2. Choose a build type: **d**=debug, **r**=release, ...
 3. `source path/ObjexxFCL/bin/Linux/GCC/64/d/setProject`
 4. `source bin/Linux/GCC/64/d/setProject`
 5. `cd src/EnergyPlus`
 6. `mak` (or `mak -j8` to run 8 in ll)
- EnergyPlus exe put in `bin/Linux/GCC/64/d`

Building EnergyPlus: Windows

1. Open a console in the repository root
 2. Choose a build type: **d**=debug, **r**=release, ...
 3. `path\ObjexxFCL\bin\Windows\GCC\64\d\setProject`
 4. `bin\Windows\GCC\64\d\setProject`
 5. `cd src/EnergyPlus`
 6. `mak` (or `mak -j8` to run 8 in ll)
- EnergyPlus exe put in `bin\Windows\GCC\64\d`

Running EnergyPlus

C++ version runs the same as the Fortran

- Input files need Linux terminators for now (will fix)

runEnergyPlus.py dev/testing script

- Runs a specified executable
- Can force annual run
- Can run under valgrind or gdb with options

Can also build/run with gprof, prof, oprofile, ...

Get Used to ...

Case sensitive identifiers and keywords
namespaces instead of modules

Header updates, include guards, ...

for loops (not exactly like DO loops)

No built-in multidimensional arrays

C++ approach to OPTIONAL and POINTER

std::string (not fixed size)

Exploding error messages (GCC is bad, Clang is good)

DO Loops v. for Loops

```
DO i = 1, N  
  N = N + 1  
END DO
```

Termination condition
computed before loop

```
// Wrong! (don't wait up)  
for ( i=1; i<=N; ++i ) {  
  ++N;  
}
```

```
// Right*  
for ( i=1, e=N; i<=e; ++i ) {  
  ++N;  
}
```

* & faster if \$\$\$ term expression

Fortran Strings

```
CHARACTER :: name(7) = "Unknown"
```

```
...
```

```
name = "Homer" ! Still 7 characters long
```

```
...
```

```
! Trailing spaces ignored in comparisons
```

```
IF ( name == "Homer" ) PRINT *, "Doh!"
```

```
! Fixed length: Quiet truncation
```

```
name = "Aristotle" ! Get Aristot
```


C++ Strings

```
std::string name( "Unknown" );
```

```
...
```

```
name = "Homer"; ! Now 5 characters long
```

```
...
```

```
// Trailing spaces matter
```

```
if ( name == "Homer" ) std::cout << "Doh!";
```

```
// Expands as needed
```

```
name = "Aristotle";
```

Data Structures

C++ Standard Library offers many choices

- Understand their uses and complexity/cost
- Choose the right one (or write your own)
- Not restricted to arrays for everything

array, vector, list, deque, map, set, pair, tuple, ...

Beware map performance for small key sets

Boost has some great containers and tools

Heap Memory Management

Proper memory/lifetime control is hard

New-to-C++ devs can break the code easily

Policies: training, heap use, smart pointers, ...

RAII/safe use within well-crafted classes

Smart pointers across boundaries

Initialization & Scope

`REAL :: x = 123.456 ! Implies SAVE (static)`

`REAL, SAVE :: x = 123.456 ! Same thing`

`float x = 123.456; // Not static`

`float x(123.456); // Not static`

`static float x(123.456); // Static`

Anti-C-ism

Devs who know C will introduce unsafe code:

- C arrays -> Buffer overflows
- char* strings
- C API for string and memory manipulation

Policy to exclude such use is a good idea

ObjexxFCL has safer C-style arrays and strings

C++ Policies: Ideas

No C-style arrays or strings

No raw pointers running wild

RAII & smart pointer plan

Private data in a real class

C++11 for loops for full container traversals

Assert for pre/post-conditions & invariants

Unit testing requirements for new/modified code

Questions

