Developing in C++ with OpenSolaris and Sun Studio 12

- GCC is the “de facto” standard compiler for Open Source Software
- OpenSolaris incorporates a number of F/OSS software components
- Default compiler on Solaris is Sun Studio
- O/N built with Studio 11, transitioning to Studio 12
- F/OSS components mixed between GCC and Sun Studio

Developing in C++ with OpenSolaris and Sun Studio 12

- Sun Studio 12 : libCstd.so.1 not compliant with C++ Standard
- Many partial template specialization features missing
- GCC/libstdc++ almost 100% Standard Compliant
- F/OSS developers are used to libstdc++
- Availability of BOOST taken for granted

Developing in C++ with OpenSolaris and Sun Studio 12

- KDE4 needs a fully compliant Standard C++ Library
- KDE4 uses BOOST [>= 1.33.1]
- Many other F/OSS components KDE4 depends on use BOOST, and make extensive use of Partial Template Specializations

Developing in C++ with OpenSolaris and Sun Studio 12

- KDE4 with libCstd.so.1 : Not Gonna Happen
- BOOST with libCstd.so.1 : Can Happen, but is insane
- Would require too many patches
- Many of KDE4's dependencies would require extensive patching as well
- These patches would probably not be accepted upstream
- Maintenance becomes a burden
- Maintainers go batty

Developing in C++ with OpenSolaris and Sun Studio 12

- Root of the problem is the Sun Standard C++ Library [libCstd.so.1]
- http://stdcxx.apache.org/
- ISO/IEC:14882:2003 compliant
- Fully supported on Solaris and Linux with Sun Studio
- Works very well with Sun Studio 12
- Thread-safe
- Full Internationalization
- 64-bit clean

Developing in C++ with OpenSolaris and Sun Studio 12

- We can now compile BOOST 1.34.1
- We are using it too!
- We can now compile everything else
- We patch less
- We complain more [because we are Standard C++ Compliant]
- Q: Why Sun Studio 12 ?
- A: [quoting Marc Mutz]: “If Studio 12 says it's wrong, it's wrong.”

Developing in C++ with OpenSolaris and Sun Studio 12

- Sun Studio12 C++ : **Atrociously Strict**
- Slightest deviation from the Standard results in an Error. You asked for it, you got it.
- Supports GCC extensions [-Qoption ccfe -features=gcc]
- Benefits:
  - Code becomes very portable
  - Code becomes very clean
  - Strictness identifies not only language violations, but also discovers design flaws

Developing in C++ with OpenSolaris and Sun Studio 12

Concrete Examples

Developing in C++ with OpenSolaris and Sun Studio 12

- Exiv2

```
namespace Exiv2 {
    typedef std::pair<uint32_t, uint32_t> URational;
    typedef std::pair<int32_t, int32_t> Rational;
}

// Let's write some code

    URational r0(std::make_pair<uint32_t, uint32_t>(3, 2));
    URational r1; // hmmm. what's in here, exactly ?
    r.first = 3;
    r.second = 2;

// Why does it have to be so ugly ?
// Do numbers actually have a “first” and a “second” ?
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Exiv2 (continued)

```
namespace Exiv2 {
    class URational : public std::pair<uint32_t, uint32_t>
    {
        public:
            URational();
            URational(const URational& rhs_);
            URational(uint32_t nom_);
            URational(uint32_t nom_, uint32_t denom_);
            ~URational();
            URational& operator= (const Exiv2::URational& rhs_);
            URational& operator= (uint32_t nom_);
            inline operator double() const
                { return double(first) / double(second); }
    };
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Exiv2 (continued)

```
// Let's write some code

namespace Silly {
    void mySillyFunction(double d_)
    {
        std::cerr << “We got a double: “ << d_ << endl;
    }

    void myOtherSillyFunction()
    {
        URational r(3, 2);
        mySillyFunction(r); // isn't this nice ?
    }
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Exiv2 (continued – the Old Way)

```
namespace Silly {
    void mySillyFunction(double d_)
    {
        std::cerr << “I got a d: “ << d_ << endl;
    }

    void myOtherSillyFunction()
    {
        URational r(std::make_pair<uint32_t, uint32_t>(3, 2));
        // The following line will not compile
        mySillyFunction(r); // a std::pair<> is not a double
    }
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Why constness mismatch between declaration and definition is not a good idea:

```
class Mismatch
{
public:
    Mismatch();
    Mismatch(int32_t x_, int32_t y_, bool z_);
    ~Mismatch();

    void doSomeMismatch (int32_t x_, int32_t y_, bool z_);

private:
    int32_t _x;
    int32_t _y;
    int32_t _z;
};
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Constness mismatch (continued)

```
Mismatch::Mismatch()
    : _x(0),
    _y(0),
    _z(false)
{ }

// the following is OK in GCC.
// it compiles with Studio 12 too. It just won't link.

Mismatch::Mismatch(const int32_t x_, const int32_t y_, const
bool z_)
    : _x(x_),
    _y(y_),
    _z(z_)
{ }
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Incomplete Types

```
class ArchiveEntryCache {
    public:
        class SubEntry {
            public:
                EntryInfo entry;
                std::map<std::string, SubEntry> entries;
                int32_t count() const;
                SubEntry();
                virtual ~SubEntry();
        };
        class RootSubEntry : public SubEntry {
            public:
                RootSubEntry() : SubEntry() { }
                bool indexed;
        };
        std::map<std::string, RootSubEntry> cache;
};
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Incomplete Types (continued)

  - The code from the example above will not compile with Sun Studio 12. It compiles with GCC.
  - Why ?
  - Let's rewrite in a portable way which compiles with both Sun Studio 12 and GCC

Developing in C++ with OpenSolaris and Sun Studio 12

- Incomplete Types (continued)

```
class ArchiveEntryCache {
    public:
    class SubEntry;
    typedef boost::shared_ptr<ArchiveEntryCache::SubEntry>
           SubEntryPtr;
    class RootSubEntry;
    typedef boost::shared_ptr<ArchiveEntryCache::RootSubEntry>
           RootSubEntryPtr;
    class SubEntry {
       public:
       EntryInfo entry;
       std::map<std::string, ArchiveEntryCache::SubEntryPtr>
entries;
       int32_t count() const;
       SubEntry() { }
       virtual ~SubEntry() { }
    };
};
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Incomplete Types (continued)

The C++ Standard Says:

ISO/IEC:14882:2003:3.9.6:

A class that has been declared but not defined, or an array of unknown size or of incomplete element type, is an incompletely-defined object type. (38). Incompletely defined object types and the void type are *incomplete types* (3.9.1). **Objects shall not be defined to have an incomplete type**.

Developing in C++ with OpenSolaris and Sun Studio 12

- Happy Constructors

```
class MyClass {
        public:
                MyClass();
                MyClass(const MyClass& rhs_);
                MyClass (const std::string& str_);
                virtual ~MyClass();
                MyClass& operator= (const MyClass& rhs_);
                void reset();

        private:
                std::string _foo;
};
```

- Let's implement it on the next slide

Developing in C++ with OpenSolaris and Sun Studio 12

- Happy Constructors (continued)

```
MyClass::MyClass()
    : _foo(“”)
    { }

MyClass::MyClass(const MyClass& rhs_)
    : _foo(rhs_._foo)
    { }

MyClass::MyClass(const std::string& str_)
    : _foo(str_)
    { }

void
MyClass::reset()
{
    _foo = std::string::string(); // VERBOTEN!!!
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Happy Constructors (continued)
- This is an assignment to a partial Constructor Definition [illegal], followed by a semicolon [illegal]:

```
_foo = std::string::string();
```

- This is an assignment to an anonymous temporary [legal] created by a call to the Default Constructor [legal]:

```
_foo = std::string();
```

Developing in C++ with OpenSolaris and Sun Studio 12

- The Ternary Operator

```
int32_t i = 3;
int32_t j = 9;
QString foo(“foo”);
QString bar;

bar = j % i ? “bar” : foo;
```

- This code will not compile with Sun Studio
- The post-condition operators of the Ternary Operator must be of the same Type.

Developing in C++ with OpenSolaris and Sun Studio 12

- Ambiguities

```
#include <algorithm>
#include <functional>
#include <cmath>
using namespace std;

int
main(int argc, char* argv[])
{
    int32_t l0 = -12;
    int32_t l1;

    l1 = std::abs(l0);
    return 0;
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Ambiguities (continued)

```
"testambig.cc", line 14: Error: Overloading ambiguity between
"std::abs(double)" and "std::abs(float)".
1 Error(s) detected.
```

- An `int32_t` can type-convert to either `float` or `double`. Hence, the call to `std::abs<T>` is ambiguous. According to the Standard.

Developing in C++ with OpenSolaris and Sun Studio 12

- We Already Have A `bool`.

```
#include <ios>
#include <iostream>
using namespace std;

#include <stdint.h>
#include <stdbool.h>

int
main(int argc, char* argv[])
{
    bool b = true;
    int32_t i = 1;

    std::cout << "b is " << std::boolalpha << b << " i == "
              << i << endl;
    return 0;
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- We Already Have A `bool` (continued).

```
"/usr/include/stdbool.h", line 42: Error: #error "Use
of<stdbool.h> is valid only in a c99 compilation
environment.".
```

- C99 is illegal in C++
- To enable some of the X/Open extensions in C++, use `-D_XOPEN_SOURCE=500 -D_XPG5`
- The corresponding defines for C99 are `-D_XOPEN_SOURCE=600 -D_XPG6`

Developing in C++ with OpenSolaris and Sun Studio 12

- INFINITY and NAN
- They don't exist in Standard C++
- We have better equivalents:

```
#include <iostream>
#include <limits>
using namespace std;

int
main(int argc, char* argv[])
{
    double myNan = std::numeric_limits<double>::quiet_NaN();
    double myInf = std::numeric_limits<double>::infinity();

    return 0;
}
```

Developing in C++ with OpenSolaris and Sun Studio 12

- Why all this
- If you are here, it's likely that you love C++
- You love C++ because it is type safe
- You love C++ because it is statically typed
- You love C++ because it is a very difficult, and powerful language
- You love C++ because of is inherent power of abstraction
- You love C++ because of its inherent intolerance to approximations

Developing in C++ with OpenSolaris and Sun Studio 12

- Why all this (continued)
- With Studio 12 you will discover C++ the way it was intended by its designers
- You wil discover that oriented programming is not a Nebulous Art
- In The Beginning, There Was C
- C is still there, even in C++
- But C++ is Better
- Your code will become squeaky clean, and portable

Developing in C++ with OpenSolaris and Sun Studio 12

- Why All This (continued)
- **Atrocious Strictness** will make you realize that OO Design and Application Design are subservient to Language Design
- If you relax the Language Design, the Application Design will develop subtle (or not-so-subtle) bugs or defects
- No other language provides such a constant feedback between its Specification and a particular application implementation

Developing in C++ with OpenSolaris and Sun Studio 12

- Why All This (continued)

- Studio 12 works on Linux too
- Free As In Beer
- You can download it from here: http://developers.sun.com/sunstudio/downloads
- KDE4 : It Builds With Sun Studio 12!