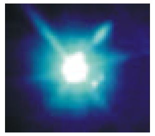
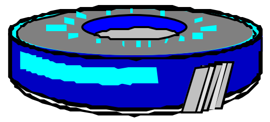


Data Access
Experiences Implementing an OO Library on Various Platforms

Ralph Lange, BESSY
Jeff Hill, LANL



Outline

- Features
- Targets
- Tests
- Performance
- Object Code Size
- Steps Taken
- Conclusion

Features

- **Extensibility.** Applications may define new containers.
- **Range Checks.** Conversion routines check data validity.
- **Type safety.** C++ features (overloaded functions) are used.
- **Multi-dimensional arrays.** Arbitrary size and number of dimensions, extraction in any size chunks are supported.
- **Improved conversion table design.** Uses templates instead of generated code.

Targets

Workstations:

- Many different compilers with different characteristics implementing different parts of the language standard.
- Fast machines with a short duty cycle: library code size and efficiency less important.

Design Rule

Portability is a must.

Other Targets

I/O Controllers:

- Realtime OS with frozen compiler version.
- Legacy systems: slow processors without virtual memory put strict demands on code size and performance.

More Design Rules

Performance is important.

Size matters.

Test Bed

Platforms:
Sun Sparc Ultra-30, Pentium PC, Motorola 68K

Compilers:
Sun WSpro, Microsoft C++, GNU g++

Performance was tested using a small any-to-any container converter. Size numbers are given for the data conversion routines on GNU/Linux.

Performance

- Callback mechanism instead of virtual functions avoids going through a function table for every call.
- Numeric data: 1us on Pentium/GNU, 4us on Sparc/WSpro.
- Sparc/WSpro: 50% longer when one of the values is unsigned, the other signed.
- Numeric from/to string: 10 times longer.
- Arrays: 1us + (0.02us ... 0.1us) per element + 0.6us per chunk on Pentium/GNU, 4 times as much on Sparc/WSpro.

Object Size

- Numbers are for Pentium/GNU, Sparc/WSpro: x2, Motorola 68k: x0.5
- Templates may create enormous amounts of object code. In this case, a conversion function with 2 formal type arguments (conversion from/to 15 basic data types) is instantiated 225 times!
- At a certain point, the conversion function object sizes had grown to 8 MB (array) resp. 5 MB (scalar).

Steps Taken

Optimize and remove debug info. Templates and inline declarations create a lot of debug information.

Move throw() into a separate class. Throw()ing is expensive: it adds several hundred bytes per instance.

Use inline and complex algorithms judiciously inside templates. Each line of code will get instantiated many times.

Use implicit conversion. The compiler will promote function arguments to wider types.

Size Reduction

Stage	Array	Scalar
Initial values	8 MB	5 MB
Optimized, no debug info	1 MB	360 KB
Without throw()	375 KB	200 KB
Without array copy code and inlines	195 KB	195 KB
Using implicit conversion	193 KB	132 KB

Conclusion

- The new interface provides important features.
- Some workarounds still necessary for compilers that don’t stick to the standards.
- Straightforward design and implementation is almost impossible: many compiler dependent implementation details have to be taken into account.
- The remaining size and performance overhead will be neglectable compare to the benefits of the new interface.

Roadmap

- Finish and test the Data Access interface on top of the IOC Database.
- Change CA server to use DA instead of GDD. *(The old CA server can go away.)*
- Change CA client to use DA instead of GDD.

- The CA Proxy Gateway will be a good test bed for the resulting new CAS and CAC user interfaces.