CPPTCL

TCL EXTENSIONS IN C++

Shannon.Noë@FlightAware.com
TCL Wiki page https://wiki.tcl.tk/13040

Thanks
CPPTCL HISTORY

Created on SourceForge 2004-11-03 by Maciej Sobczak

When C++ was a pain

FlightAware started working with it in 2017

- Moved to github
  https://github.com/flightaware/cpptcl
- Added some enhancements
- Implemented documentation as markdown
TECHNICAL STUFF
WHAT ARE TCL EXTENSIONS?

TCL extensions add new commands to TCL interpreters with C
DYNAMIC LOADING

making new programs at runtime

• Break a program into smaller parts.
• Re-assemble the parts at runtime.
• Combine different parts to make new programs at runtime.
WHAT IS A TCL EXTENSION?

- native platform instructions - compiled code
- in a shared library - a special file
- that call the TCL C API to extend the TCL interpreter
- implemented using dynamic loading
USING C++ TO EXTEND TCL
RATIONALE

• Addresses increasing performance problems.
  ▪ C++ is close to the OS
  ▪ C++ memory management can be explicit
  ▪ C++ includes all multi-core techniques
• Provides high quality development platform.
  ▪ clang and libc++ are moving C++ rapidly
  ▪ Xcode
  ▪ valgrind
MODERN C++

C++98
- Templates
- STL including containers and the algorithms
- Strings
- I/O Streams

C++11
- Move semantic
- Unified initialization
- auto and decltype
- Lambda functions
- constexpr
- Multithreading and the memory model
- Regular expressions
- Smart pointers
- Hash tables
- std::array

C++14
- Reader-writer locks
- Generalized lambda functions

C++17
- Fold expressions
- constexpr if
- Structured binding declarations
- std::string_view
- Parallel algorithm of the STL
- The filesystem library
- std::any, std::optional, and std::variant

credit: www.modernescpp.com
PERFORMANCE PROCESS

- Profile: Find the hotspot in a program.
- Review: Check the TCL for simple performance errors.
- Rewrite: Replace TCL code with C++ as needed.
  - Sometimes with a TCL extension.
  - Sometimes replace program entirely.
Example C++ code and TCL code

```cpp
#include <cpptcl/cpptcl.h>

std::string hello(std::string name) {
    std::string r("hello ");
    r.append(name);
    return r;
}

CPPTCL_MODULE(Hello, i) {
    i.def("hello", hello, Tcl::usage("hello <string>"));
}
```

```
$ tclsh
% load ./libhello.so
% hello joe
hello joe
```
Of course you pick your build system of choice
C++ community is mostly CMake *
HOW CPPTCL WORKS

- Code generation is C++ templates
- Supports functions and methods of classes
- Supports zero to nine parameters
- Types: bool, int, long, double
- String types: std::string, char const *, std::vector<char>
- C++ classes (pointers)
class Person
{
public:
    Person(std::string const &n) : name(n) {}

    void setName(std::string const &n) { name = n; }
    std::string getName() { return name; }

private:
    std::string name;
};
CPPTCL_MODULE(Person, i)
{
    i.class_<Person>("Person", Tcl::init<std::string const &>());
    .def("setName", &Person::setName)
    .def("getName", &Person::getName);
}
% load libperson.so
% set p [Person "Joe"]
p0x55f280c7e650
% $p getName
Joe
% $p setName Mary
% $p getName
Mary
CPPTCL_MODULE C MACRO

CPPTCL_MODULE(NAME, INTERPRETER_VAR)
// which generates the extension
// C entrypoint
extern "C" {
  void ${NAME}_Init(Tcl_Interp *) {
    Tcl::interpreter ${INTERPRETER_VAR};
    ...
  }
}
using namespace Tcl;

void helloArray(object const &name, object const &address) {
  cout << "Hello C++/Tcl! from array " <<
      name("first").get() << " " << name("last");

  cout << "exists zip? " << address.exists("zip");

  std::string state("state");
  // Check for exists with if
  if (address(state)) {
    cout << "state " << address(state).asString();
  }
}
C++ CALLING TCL

```cpp
i.eval(R"(proc strcat2 {arg1 arg2} { return "$arg1+$arg2" })")
Tcl::Bind<string, string, string> strcat2("strcat2");
string val2 = strcat2("The", "End");
```