

# rootJS - results of the implementation phase

Software Engineering Practice - Winter Term 2015/16

C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart | February 14, 2016

STEINBUCH CENTER FOR COMPUTING

```
NodeHandler::exposeGlobals() throw (std::invalid_argument)
{
    Collection *globals = gROOT->GetListOfGlobals();
    TIter next(globals);
    v8::Local<v8::Object> exportsLocal = v8::Local<v8::Object>::New(v8::Isolate::GetIsolate(),
    while (TGlobal *global = (TGlobal*) next())
    {
        if (!global->IsValid() || (global->GetAddress() == nullptr))
        {
            Toolbox::logInfo("Invalid global instance found.");
            continue;
        }
    }
}
```

- 1 Fullfilled criteria
  - Required criteria
  - Limiting criteria
- 2 Changes in design
  - AsyncRunner
  - MetaInfo
  - Other changes
- 3 Workflow
- 4 Statistics

The bindings must:

- Work on Linux
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter
- Accept C++ code for just-in-time compilation
- Update dynamically following changes to C++ internals
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

The bindings must:

- Work on Linux ✓
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter
- Accept C++ code for just-in-time compilation
- Update dynamically following changes to C++ internals
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

The bindings must:

- Work on Linux ✓
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter ✓
- Accept C++ code for just-in-time compilation
- Update dynamically following changes to C++ internals
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

The bindings must:

- Work on Linux ✓
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter ✓
- Accept C++ code for just-in-time compilation ✓
- Update dynamically following changes to C++ internals
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

The bindings must:

- Work on Linux ✓
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter ✓
- Accept C++ code for just-in-time compilation ✓
- Update dynamically following changes to C++ internals ✓
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

The bindings must:

- Work on Linux ✓
- Allow the user to interact with any ROOT class from the Node.js JavaScript interpreter ✓
- Accept C++ code for just-in-time compilation ✓
- Update dynamically following changes to C++ internals ✓
- Provide asynchronous wrappers for common I/O operations (i.e. file and tree access) ✓



The bindings should not:

- Add any extending functionality to the existing ROOT framework
- Necessarily support previous/future ROOT versions

The bindings should not:

- Add any extending functionality to the existing ROOT framework ✓
- Necessarily support previous/future ROOT versions

The bindings should not:

- Add any extending functionality to the existing ROOT framework ✓
- Necessarily support previous/future ROOT versions ✓

- We added a helper class for asynchronous operations, 'AsyncRunner'
- Uses libuv internally
- TThread doesn't allow communication between node threads

- We added a helper class to encapsulate differences between TGlobals and TDataMember
- TGlobal has a getAddress() method to get the absolute adress, TDataMember only getOffset()

- Dynamic library loader: ROOT doesn't load enough libraries on its own
- removed get exports

- Test-driven development
- We couldn't use Travis CI, the build of root always timed out
- So we used Jenkins instead
- Implement new features in branches first, then merge later

- 3200+ LOC
- 400+ commits