Karlsruhe Institute of Technology

Software Engineering Practice

WINTER TERM 2015/2016

rootJS - Implementation Report

Node.js bindings for ROOT 6

Jonas Schwabe
Theo Beffart
Sachin Rajgopal
Christoph Wolff
Christoph Haas
Maximilian Früh

supervised by
Dr. Marek Szuba

Contents

1. Introduction

1.1. About rootJS

The purpose of creating Node.js bindings of ROOT, called rootJS, is to enable users to integrate ROOT in Node.js programs, such as Node.js based web servers.

2. Architecture

2.1. The class diagrams in comparison

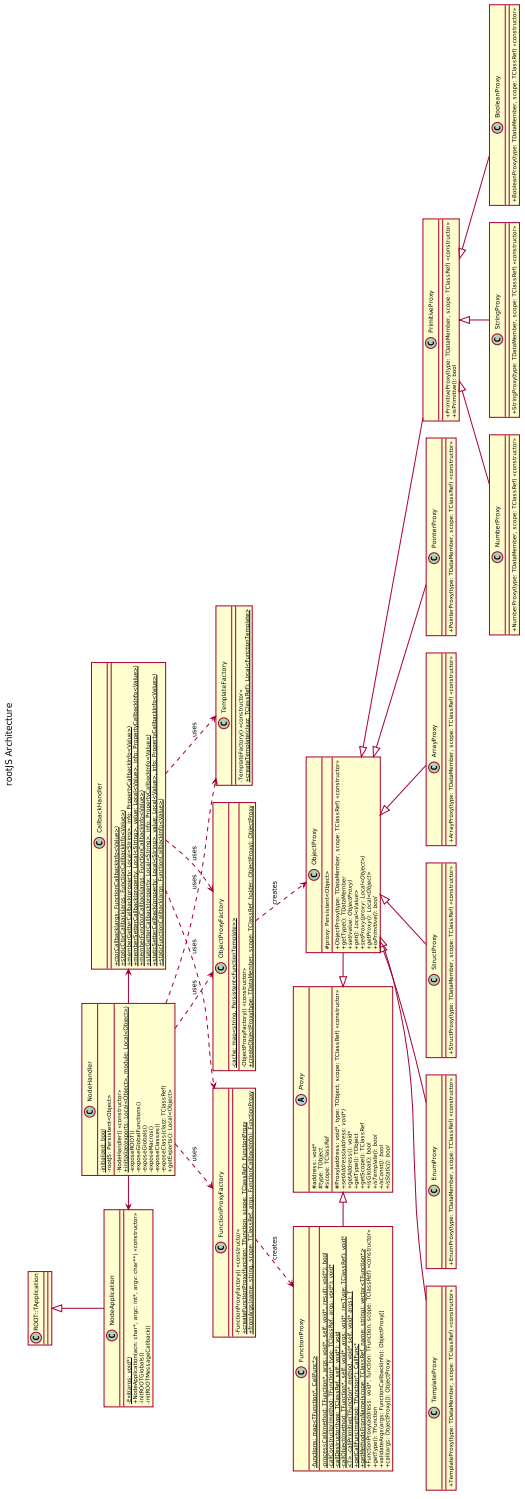

Figure 2.1: The rootJS class diagram at the conclusion of the design phase

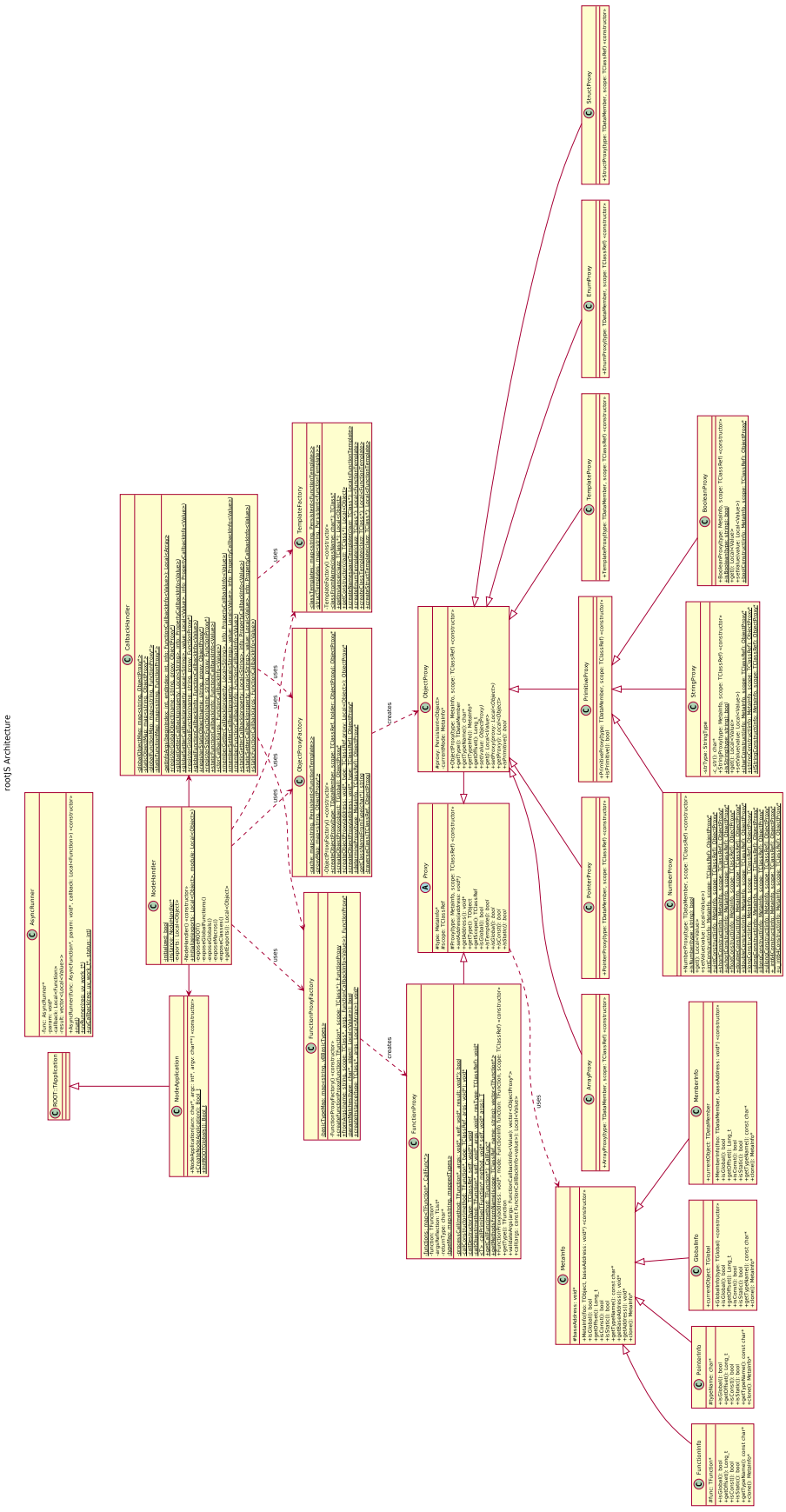

Figure 2.2: The current rootJS class diagram

2.2. The Changes in the Architecture in detail

- MetaInfo
- AsyncRunner
- TemplateFactory
- FunctionProxyFactory
- clone() and backup()
- ObjectProxyFactory
- "datatype"construct methods

3. Criteria

3.1. Required Criteria

These are the Required Criteria as stated in the functional specification:

- work on Linux

This is rather straightforward as long as the user has installed the required dependencies. This means that, at the very least, ROOT and Node.js have to be installed. ROOT is supported on only a few major Linux distributions, so rootJS is limited to the systems on which ROOT has been installed.

- allow the user to interact with any ROOT class from the Node.js JavaScript interpreter

The user is able to interact with any ROOT class by requesting any class with rootJS. The following is an extract from the description of the NodeHandler in the module guide, which describes how rootJS is started up:

```
// JavaScript: Load ROOT bindings in JavaScript
var root = require(rootJS.node);

// C++: Expose the initialize method as the main entry point
NODE_MODULE(rootJS, initialize)
```

A file *index.js* is offered to make initializing rootJS easier for the user which will also handle user interface callbacks.

index.js :

```
module.exports = require('./build/Release/rootjs.node');

var uicallback = function() {
module.exports.gSystem.ProcessEvents();
setTimeout(uicallback, 100);
}
uicallback();
```

If the user would then like to open a TBrowser() from ROOT, then they would have to do the following:

```
var root = require('./index.js');
new root.TBrowser();
```

The user would then have access to a TBrowser() from ROOT.

- accept C++ code for just-in-time compilation

The user will have access to the Node.js interpreter and can enter single lines of code in it. These lines of code will be interpreted by rootJS and rootJS will call Cling to interpret them.

- update dynamically following changes to C++ internals

- provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

While no specifics were written about asynchronous processes in the module guide, two variations were considered. The first variation would utilize ROOT's inbuilt multi-threading environment. ROOT contains a TThread class which is similar to the std::thread class in C++, but is adjusted to ROOT. The second variation would use libuv, a software library that provides asynchronous even notification and was mainly designed for Node.js.

3.2. Optional Criteria

These are the Optional Criteria as stated in the functional specification:

- support the streaming of data in JavaScript Object Notation (JSON) format compatible with JavaScript ROOT
- implement a web server based on Node.js to mimic the function of the ROOT HTTP server
- work OS independent (i.e. support Mac OS X, Linux operating systems)

3.3. Limiting Criteria

These are the Limiting Criteria as stated in the functional specification:

- add any extending functionality to the existing ROOT framework

No code was written to implement any new features which are not already existing in the ROOT framework.

- necessarily support previous ROOT versions

ROOT 6 uses the LLVM-based C++ interpreter Cling, while ROOT 5.34 uses the C++ interpreter CINT. rootJS was designed with ROOT 6 in mind and many of the function calls, etc. will only work with Cling. As of yet, rootJS has not been tested not been test on older ROOT versions, but it is very unlikely that rootJS will function with ROOT 5.34.

- necessarily support future ROOT versions

If future versions of ROOT also utilize Cling, rootJS might be able to support them. Therefore it is likely that rootJS should support all versions of ROOT 6.

4. Unit Tests

4.1. C++ Unit Tests

Initially, the aim was to have unit tests running for the C++ code written for rootJS. Google Test was chosen for unit testing. However, an insurmountable issue appeared. It was not possible to make Google Test work with rootJS, because all files included in the rootJS source files also had to be built. This includes ROOT and Google v8. ROOT was built just fine, but it was not possible to build v8. This was not an issue with Google Test, but an issue with v8 itself. There may have been a possibility to build v8, but there little to no documentation about how to do it, and would have taken up a considerable amount of time. Consequently, rootJS has no unit testing.

4.2. JavaScript Tests

While there are no C++ tests, it is possible to gain significant coverage by using JavaScript Tests. Mocha is the JavaScript testing framework used for rootJS. Mocha runs on Node.js and explicitly supports asynchronous testing, meaning it is ideal for testing rootJS' asynchronous capabilities.

Miscellaneous

- should be possible to have non pointer references to returned objects from function calls

Functions overloading

- should not be possible to call Printf without the corect args
- should return a number when calling Hash on a string
- should be possible to call a function that returns a char*
- should be possible to call a function with a callback that is called when the root code has been processed

Interface require

- should return an object

Globals/numbers

- should have a property named kMaxChar
- should have kMaxChar which equals 128
- should save gDebug as an integer (drop decimal values)
- should overflow silently
- should not accept values that are not numeric
- should have a property named kInitTreatAsZero
- should have kInitTreatAsZero which equals 1e-12
- should have kInitTreatAsZero which equals 1e-12
- should fail to write to const globals strings
- should have the gProgName "node"
- should consider char pointers to be immutable

Booleans

- should have kTRUE which should be true
- should have kFALSE which should be false
- should fail to change the value of kFALSE

Functions test

- should not be possible to call Printf without the corect args
- should not be possible to call gSystem.Dump without the corect args
- should have gSystem.ProcessEvents returning a boolean value
- should be possible to call virtual methods
- should be possible to call methods asynchronously
- should be possible to get return values as a param of a callback function
- should be possible to run multiple functions asynchronously in parallel

Constructor

- should be possible to create a new TString

Function return values

- boolean values should be returned even though they are false