RootJS: Node.js Bindings for ROOT 6

Theo Beffart, Maximilian Früh, Christoph Haas, Sachin Rajgopal, Jonas Schwabe, Christoph Wolff, Marek Szuba
ROOT — scientific software framework from CERN
- *de facto* standard tool in particle physics
- written mainly in C++
- essentially a library of classes
- integrated C++ interpreter based on LLVM (ROOT 6)
- bindings exist for Python, R, Ruby, …

Node.js — high-performance runtime environment for Web applications
- user code in JavaScript
- internals in C++ — Google V8 engine
- non-blocking, asynchronous I/O
- popular in modern large-scale systems

Web applications in high-energy physics
- numerous — *e.g.* data browsers, quality assurance, …
- back-end processing with ROOT frequently desired
Project Overview

- **RootJS: Node.js bindings for ROOT 6**
  - JavaScript access to all ROOT classes
  - inject, JIT-compile and execute C++ macros
  - dynamically updated state
  - asynchronous

- Native Node.js module written in C++

- **System requirements:**
  - OS supporting ROOT 6
    - tested under Linux, Mac OS X
  - Node.js 4–6
    - v4 LTS recommended
Use-case Examples

Live event viewer

- Important for monitoring, basic QA
- Typically, standalone ROOT application
- Limited portability:
  - needs ROOT and dependencies installed
  - requires access to data source
- Alternative — modern Web application:
  - back-end processing: ROOT, close to detector
  - front-end anywhere in the world
  - only require a Web browser

Analytics in Web applications

- Take advantage of ROOT in machine learning, statistical analysis, linear algebra, ...
- Gain access to XRootD, PROOF etc.
Use-case Examples

**Live event viewer**
- Important for monitoring, basic QA
- Typically, standalone ROOT application
- Limited portability:
  - needs ROOT and dependencies installed
  - requires access to data source
- Alternative — modern Web application:
  - back-end processing: ROOT, close to detector
  - front-end *anywhere in the world*
  - only require a Web browser

**Analytics in Web applications**
- Take advantage of ROOT in machine learning, statistical analysis, linear algebra, ...
- Gain access to XRootD, PROOF etc.
Related Work

JavaScript ROOT

JSROOT
- Web client
- reimplementation
- feature set limited by design:
  - ROOT graphics
  - basic file input
  - THttpServer interface

RootJS
- standalone, or Web server
- wrapper
- all features of ROOT 6

Highly complementary!
- Example: Web application with RootJS back-end and JSROOT front-end.
Related Work
Jupyter Notebook

ROOT in Jupyter
- Web client and server
- sophisticated interactive interface
- not meant for scripting/batch
- limited access to third-party libraries
  - but: Python 2 kernel + PyROOT + ...

- Some overlap
- Different context — interactive vs scripted

RootJS
- standalone, or Web server
- no interactive interface (yet)
- aimed at scripted/batch processing
- full access to the Node.js ecosystem

- Jupyter + Javascript kernel + RootJS + ...
Related Work
Jupyter Notebook

ROOT in Jupyter
- Web client and server
- sophisticated interactive interface
- not meant for scripting/batch
- limited access to third-party libraries
  - but: Python 2 kernel + PyROOT + ... 
- Some overlap
- Different context — interactive vs scripted
- Jupyter + Javascript kernel + RootJS + ...

RootJS
- standalone, or Web server
- no interactive interface (yet)
- aimed at scripted/batch processing
- full access to the Node.js ecosystem

Some overlap
Different context — interactive vs scripted
Jupyter + Javascript kernel + RootJS + ...
Design Overview

- Basic architecture requirements:
  - dynamic object creation and encapsulation
  - non-blocking function calls via callbacks

- Fundamental language differences
  - different type systems (dynamic vs static)
  - a functional language
  - classless objects, prototype functions

- “Need an adapter”
  - software design pattern
  - help incompatible interfaces work together

- Environment:
  - V8 API — object exposure and callback handling
  - ROOT RTTI interface — class, namespace, global and member variable information
Fulfilment of Requirements

- provide async call context before forwarding to RTTI API
- recursively seek & expose classes and namespaces
- direct access to C++ objects in memory via corresponding proxy object
- forward constructor calls
- encapsulate constructed objects for exposure
- entry point for client interactions with ROOT

Diagram:
- init
- callback handling
- function calls
- object access
- object creation

JavaScript object
Proxy object
C++ object
Core Architecture

- init
- callback handling
- function calls
- object access
- object creation

FunctionProxyFactory
+ fromArgs()

FunctionProxy
+ call()

ObjectProxy
+ readValue()
+ writeValue()

ObjectProxyFactory
+ createCapsule()

ObjectProxy
- address

TemplateFactory
+ createTmplt()
Core Architecture

- NodeHandler
  - + exposeROOT()

- CallbackHandler
  - + onAccess()

- FunctionProxyFactory
  - + fromArgs()

- ObjectProxyFactory
  - + createCapsule()

- TemplateFactory
  - + createTmplt()

- Proxy
  - - address

- FunctionProxy
  - + call()

- ObjectProxy
  - + read() / write()

- PrimitiveProxy
Non-blocking Execution

- Killer feature of Node.js
- RootJS: can add a callback to *any* exposed function
- Asynchronous workers
- Worker–V8 message passing using libuv
  - integrates well with Node.js
  - no need to actively wait for threads
  - based on signals — non-blocking, no waste of CPU time
Installation and Invocation

- We are on npmjs.com — easy!
  
  
  
  npm install rootjs

- Requirements: Node.js, npm, C++ compiler, ROOT 6, libuv headers

- Loaded the standard Node.js way:
  
  ```javascript
  var root = require('rootjs');
  ```

- All ROOT variables and classes now accessible through `root`

- Need more libraries? No problem:
  
  ```javascript
  root.loadlibrary("libMathCore.so"); // load it...
  root.ROOT.Math.Pi(); // ...and use it!
  ```

- Callback example:
  
  ```javascript
  root.TFile.Open("foo.root", function (fin) {
    fin.ls();
  });
  ```
Infrastructure

- **Source-code repository:** Git
  - primary: GitHub
  - mirror: CERN GitLab
  - currently: \(\approx 4000\) lines of code

- **Issue tracker:** GitHub

- **Continuous integration:** CERN Jenkins
  - integration tests
  - code coverage
  - Doxygen documentation
  - currently: 90 tests, 76% line coverage
THANK YOU

https://github.com/rootjs/rootjs
https://www.npmjs.com/package/rootjs