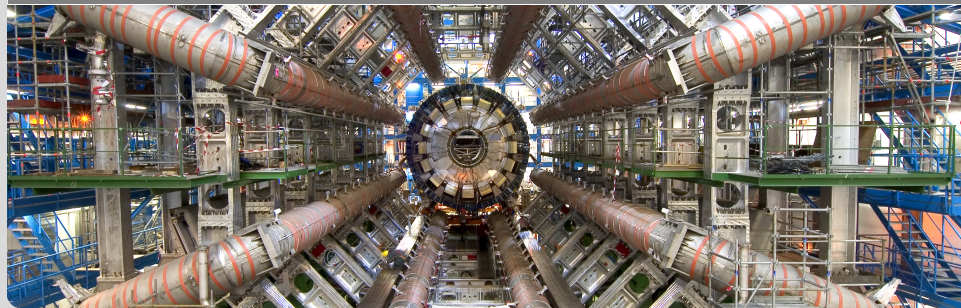


rootJS - Functional Specification

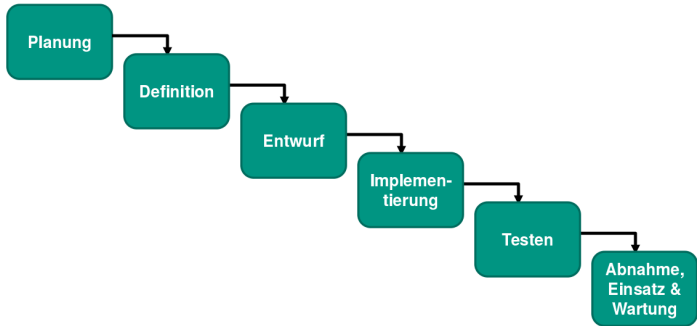
PSE - Software Engineering Practice

C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart | December 15, 2015

STEINBUCH CENTER FOR COMPUTING



- Waterfall model
 - Planning/definition



- be able to write ROOT code in Node.js programs
- integrate ROOT into Node.js based web applications

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

- 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)

The bindings should not

- add any extending functionality to the existing ROOT framework
- necessarily support previous/future ROOT versions

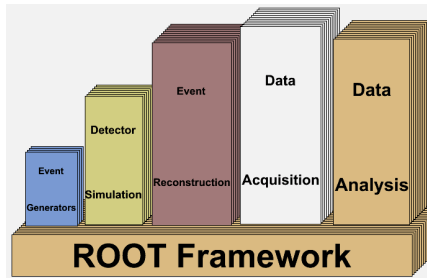
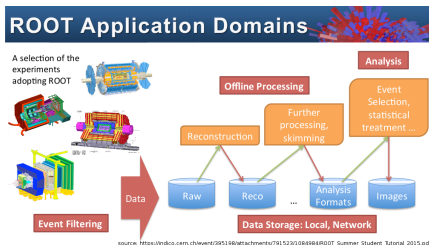
rootJS will be used to create web-applications that can:

- Expose processed data (that might otherwise be hard to access) and then visualize it locally
- Interact with data both stored somewhere accessible for the server or streamed via remote procedure call (RPC)
- Run on any platform that supports a browser

Most users of rootJS will be used to working in Linux and with web servers. At the very least, they will be able to install ROOT and also be proficient in programming languages like JavaScript and C++.

- Scientists (e.g. particle physicists)
- Researchers
- Web-developers interested in creating applications based on ROOT

- process and visualize large amounts of scientific data (CERN)
- features a C++ interpreter (CLING) - i.e. used for rapid and efficient prototyping
- persistency mechanism for C++ objects



- 

Node.js

- open source runtime environment
 - develop server side web applications
 - act as a stand alone web server
- Google V8 engine to execute JavaScript code

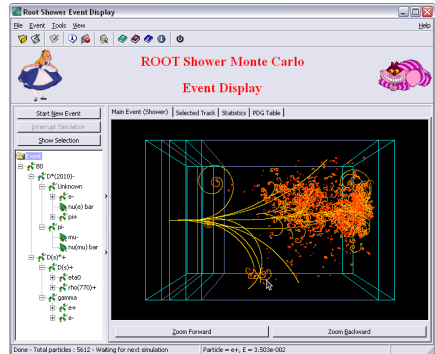
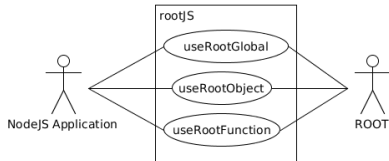


- open source runtime environment
 - develop server side web applications
 - act as a stand alone web server
- Google V8 engine to execute JavaScript code
- rootJS bindings realized as native Node.js module written in C++



- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ | ≡ ↺ 🔍 ↻

- ⇒ generally negligible hardware requirements of the bindings themselves



Flow of events

- 1 The NodeJSApplication requests access to a global variable of ROOT.
- 2 rootJS sends a request to the corresponding ROOT variable.
- 3 ROOT returns the requested variable value.
- 4 The value is passed from rootJS to the NodeJSApplication.

- ◀ ◻ ▶ ◀ ◼ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ || ≡ ↺ 🔍 ↻

| | |
|------------------------|---|
| <i>Entry condition</i> | rootJS has been initialized. |
| <i>Exit condition</i> | The value has been returned to the NodeJSApplication. |

| | |
|--------------------------------------|--|
| <i>Use case name</i> | UseROOTObject |
| <i>Participating actor instances</i> | Initiated by NodeJSApplication; Processed by rootJS, ProxyObject; Communicates with ROOT |
| <i>Flow of events</i> | |

- 1 The NodeJSApplication requests access to a ROOT object by calling a constructor function.
- 2 rootJS encapsulates the requested ROOT object within a ProxyObject that was created recursively.

Flow of events

- ③ rootJS stores the created ProxyObject in a cache memory.
- ④ The ProxyObject is exposed to the NodeJSApplication.

Entry condition

rootJS has been initialized.

Exit condition

The reference of the ProxyObject has been return to the NodeJSApplication.

Flow of events

- ④ rootJS encapsulates the returned ROOT object within a ProxyObject.
- ⑤ The ProxyObject is exposed to the NodeJSApplication.

| | |
|------------------------|--|
| <i>Entry condition</i> | rootJS has been initialized. |
| <i>Exit condition</i> | The reference of the ProxyObject has been return to the NodeJSApplication. |

Flow of events

- 1 The NodeJSApplication wants to execute ROOT specific C++ code (given as string) during runtime.
- 2 rootJS forwards the instructions to Cling.
- 3 Cling evaluates the received instructions using JIT compilation concepts and dynamically modifies the state of ROOT.

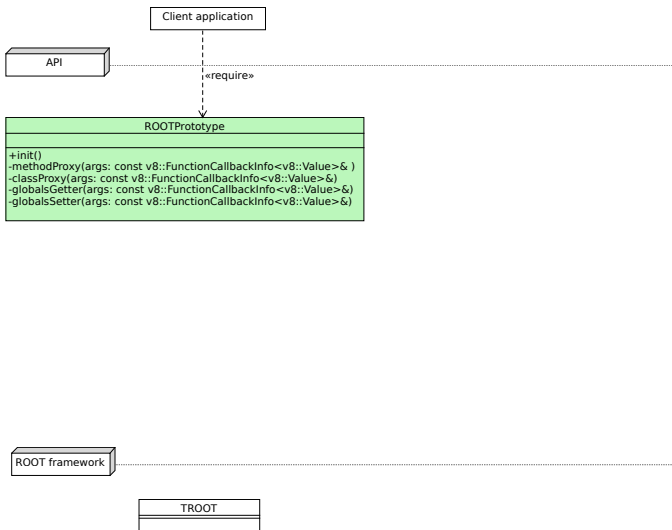
Flow of events

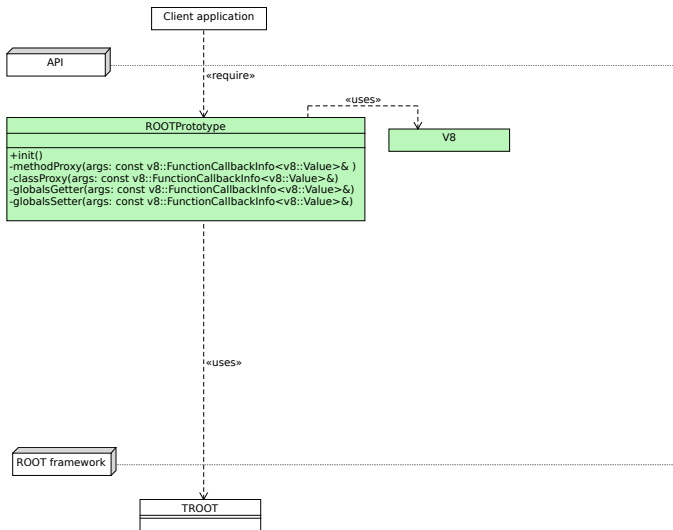
3. rootJS takes care of encapsulating exceptions possibly thrown by `Cling` or `ROOT` during evaluation and execution.
4. rootJS provides the evaluation results and corresponding return values to the `NodeJSApplication`.

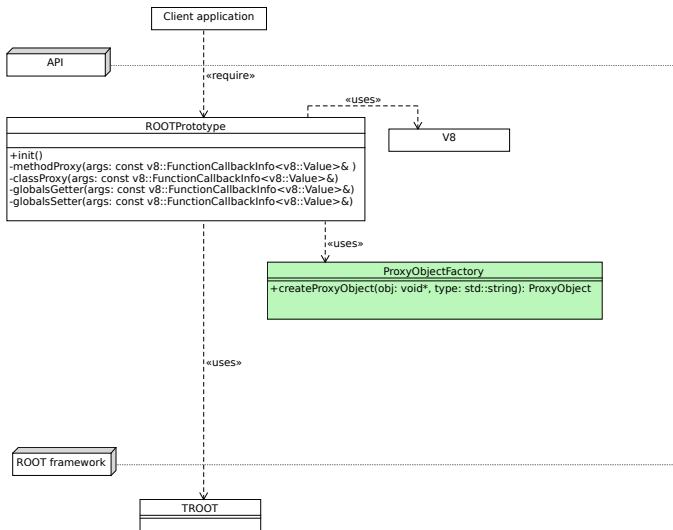
| | |
|------------------------|---|
| <i>Entry condition</i> | rootJS and Cling have been initialized. |
| <i>Exit condition</i> | rootJS either confirms the proper execution of the specified instructions or forwards thrown exceptions to the NodeJSApplication. |

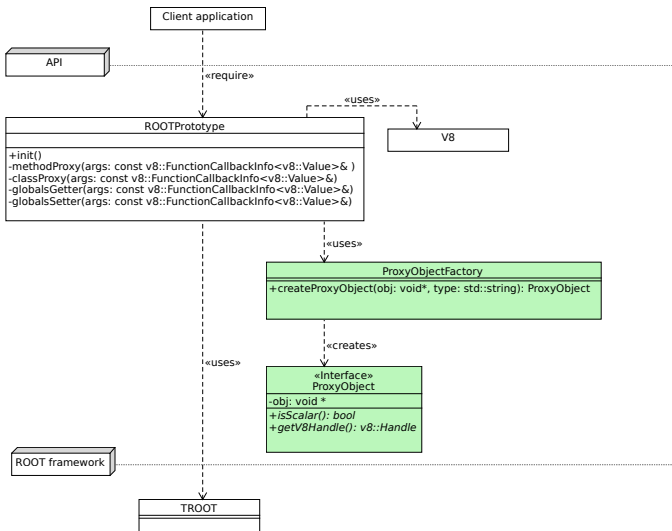
Basic Architecture



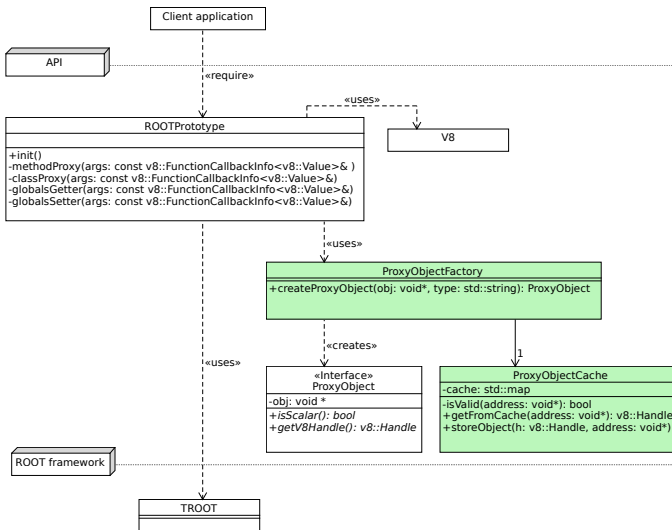


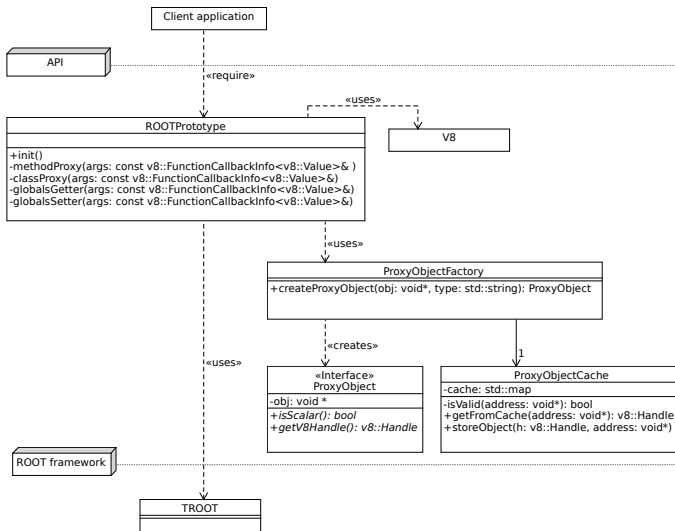






Basic Architecture

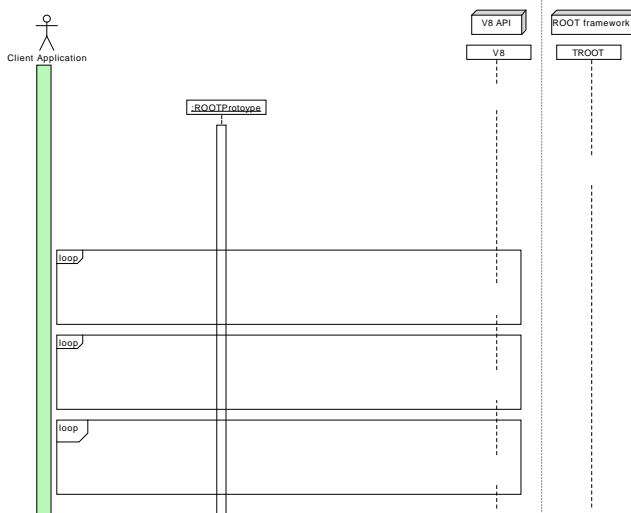


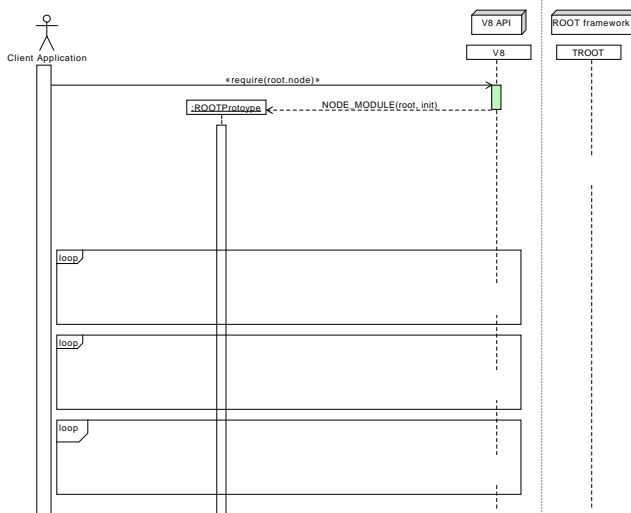


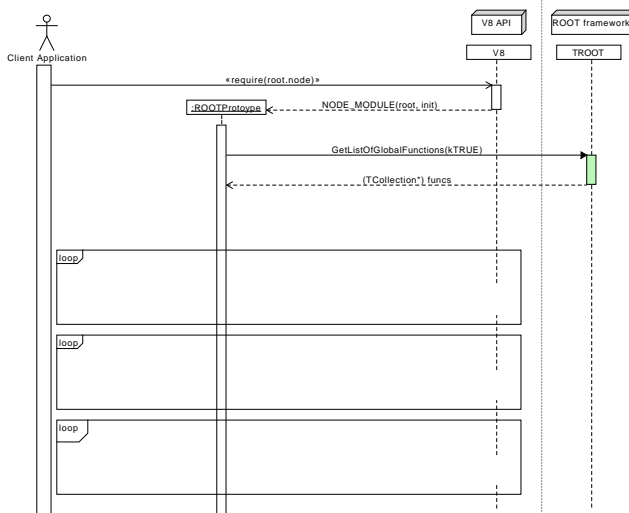
- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ || ≡ ↺ 🔍 ↻

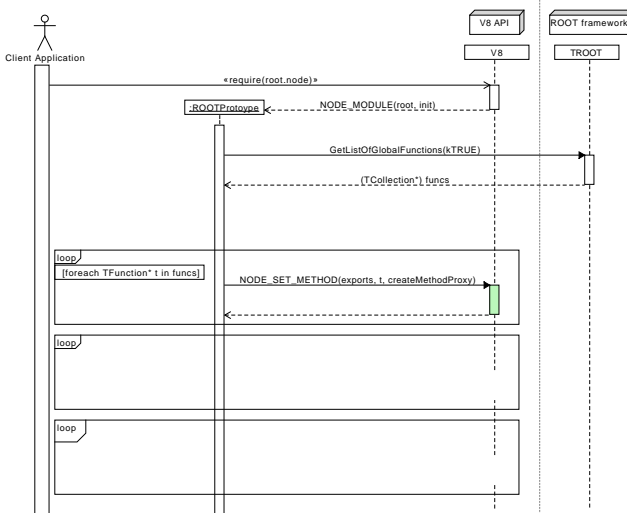
- Expose all
 - Global variables
 - Global functions
 - Classes
- Each are bound to corresponding proxy methods
- An object which members are the exposed features is being passed to node

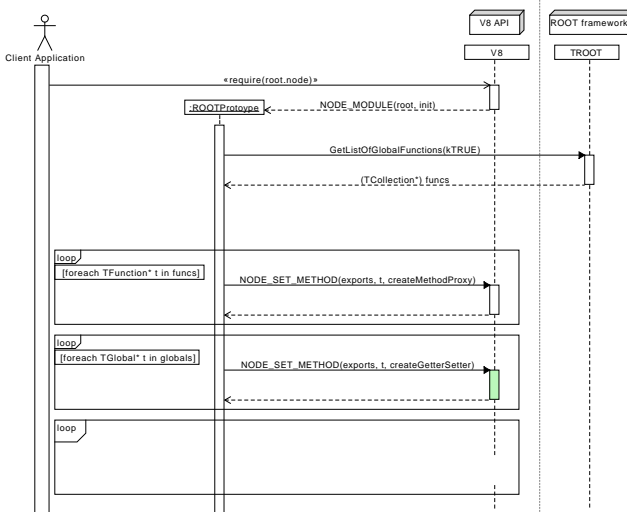
rootJS init

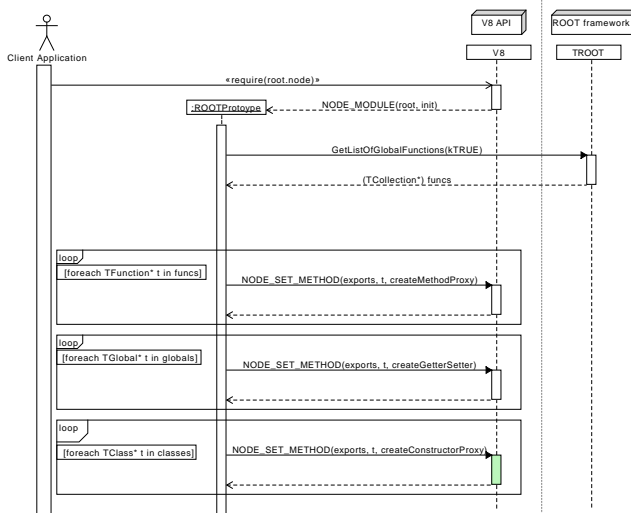




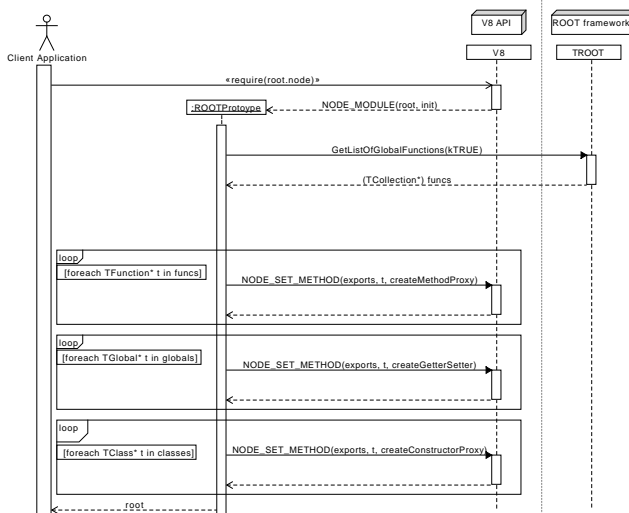








rootJS init

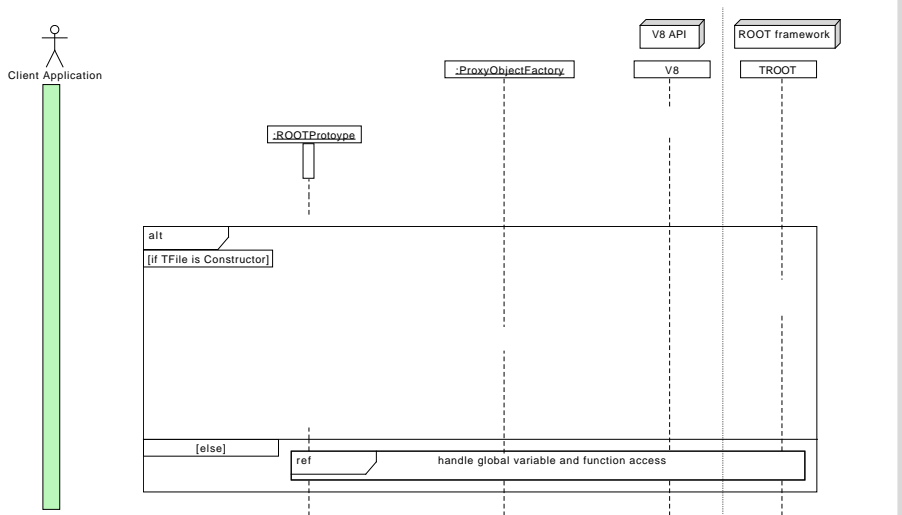


- All features in node are mapped to a proxy method that will be called

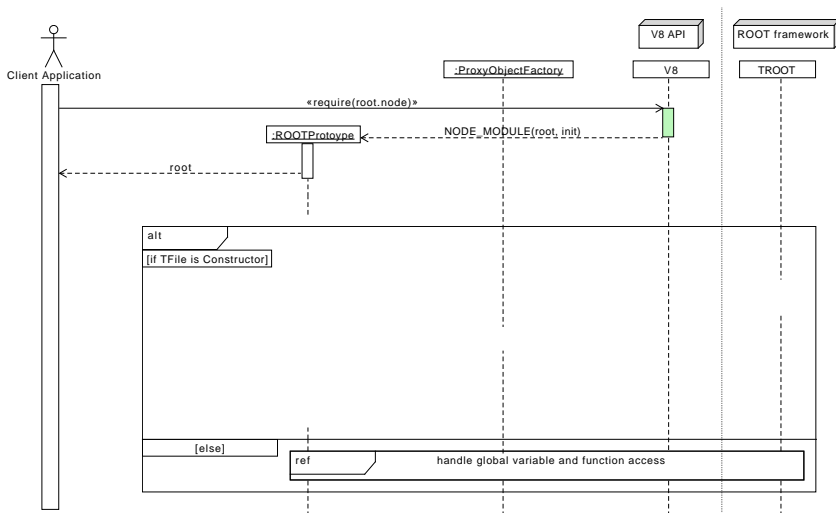
- All features in node are mapped to a proxy method that will be called
- The proxy method will eventually call a root function and pass the result to our ObjectFactory

- All features in node are mapped to a proxy method that will be called
- The proxy method will eventually call a root function and pass the result to our ObjectFactory
- By looking at the object type an corresponding v8::Handle will be generated and returned to node
 - If the result is an object this will be done recursively

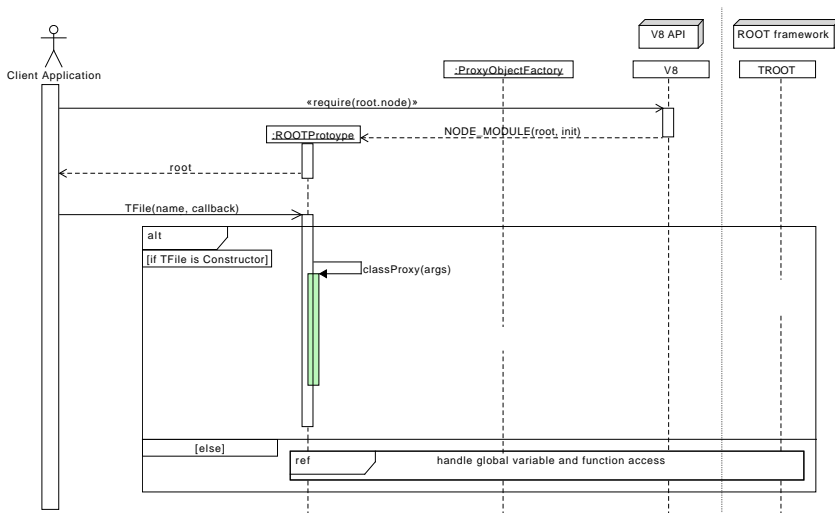
proxied file access



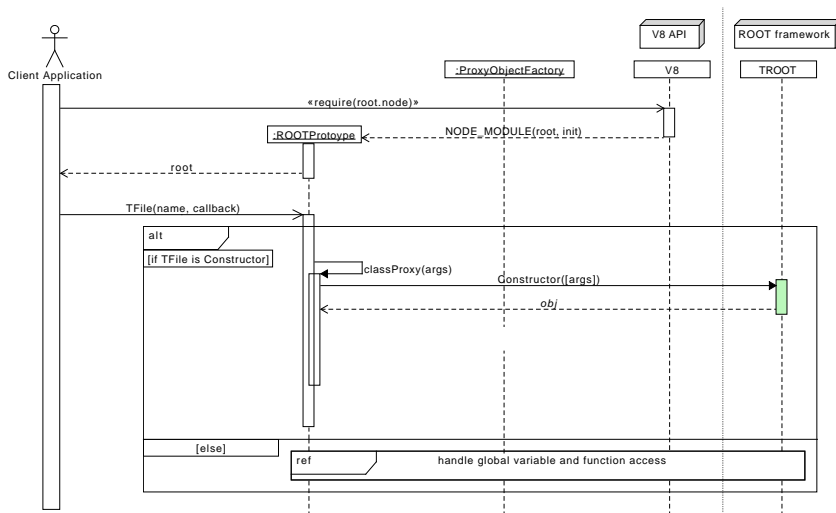
proxied file access



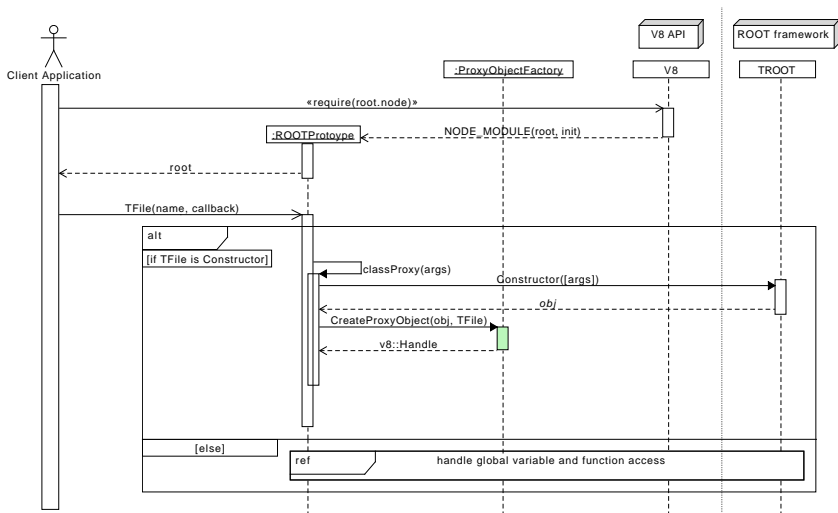
proxied file access



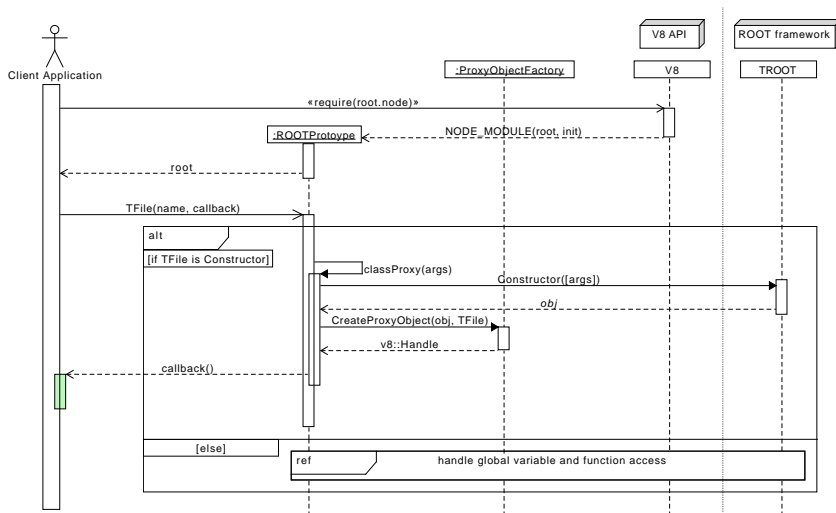
proxied file access



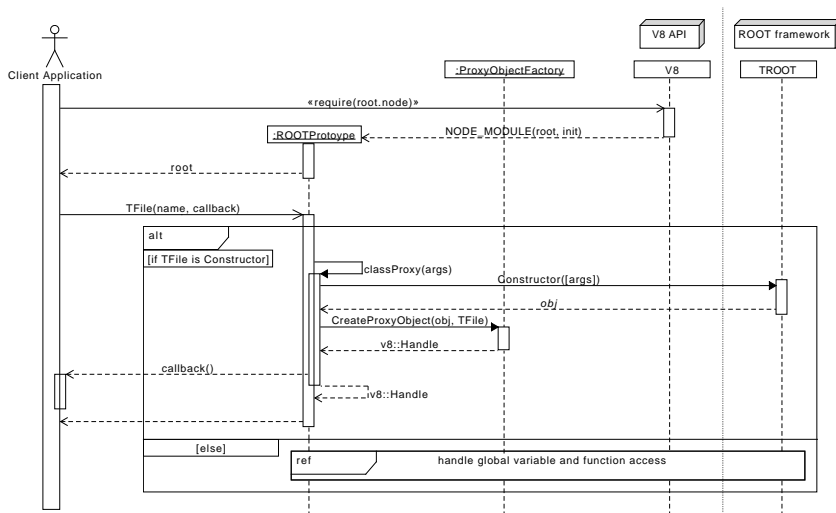
proxied file access



proxied file access

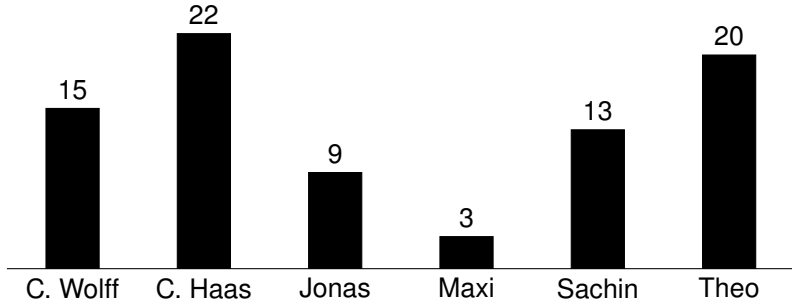


proxied file access

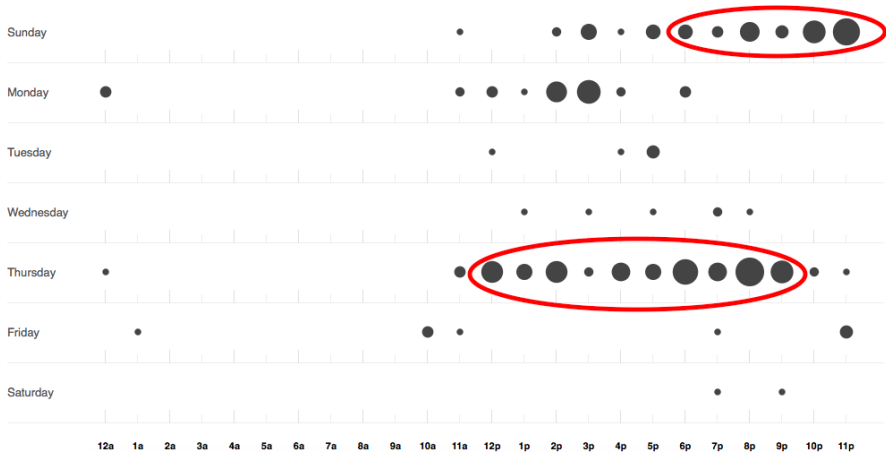







- Make sure all elements are callable without crashing
- To verify results of function calls and calculations, we would need to run ROOT's testcases
- Porting all ROOT testcases would make no sense!
- Only port a subset to make sure the bindings are working and leave the rest to the ROOT developers

Merges



Punchcard



-  CERN. *ROOT application domains*. Dec. 2015. URL: <https://root.cern.ch/application-domains>.
-  CERN. *ROOT Shower Event Display*. Dec. 2015. URL: <https://root.cern.ch/rootshower00png>.
-  exortech. *v8 logo*. Dec. 2015. URL: https://github.com/exortech/presentations/blob/master/promise_of_node/img/v8.png.
-  *Node.js logo*. Dec. 2015. URL: <https://nodejs.org/static/images/logos/nodejs-light.eps>.
-  Danilo Piparo and Olivier Couet. *ROOT Tutorial for Summer Students*. Dec. 2015. URL: https://indico.cern.ch/event/395198/attachments/791523/1084984/ROOT_Summer_Student_Tutorial_2015.pdf.



Walter Tichy. *Wasserfall Modell*. 2015.



Boris Vacher. *npm logo*. Dec. 2015. URL:
<https://commons.wikimedia.org/wiki/File:Npm-logo.svg>.