

rootJS - Functional Specification

PSE - Software Engineering Practice

C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart | December 16, 2015

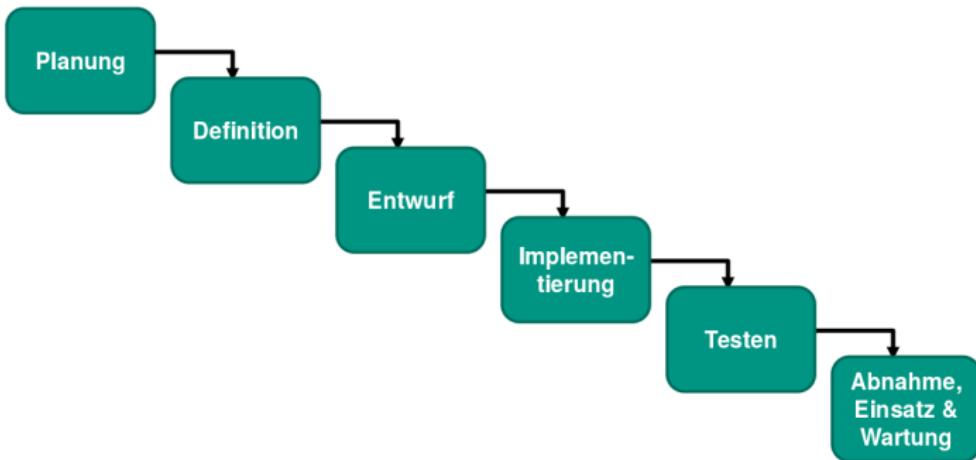
STEINBUCH CENTER FOR COMPUTING



About PSE

Praxis der Softwareentwicklung(PSE) = Software Engineering Practice

- Waterfall model
 - Planning/definition



Purpose

Node.js bindings for ROOT

- Be able to write ROOT code in Node.js programs
- Integrate ROOT into Node.js based web applications

Required Criteria

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)

Optional Criteria

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)

Limiting criteria

The bindings should not

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

Product usage

- It's JavaScript → web-applications
- Expose processed data and then visualize it locally
- Interact with remote data (i.e. streamed via RPC)
- Accessible on 'unconventional' devices (mobile phones/tablets)

Audience

- Scientists (e.g. particle physicists) and Researchers
- Typical user will know ROOT and JavaScript → rather technology proficient
- Web-developers

Operating conditions

- Servers that run ROOT
- ROOT6 is currently only available on Mac and Linux, so that's our focus

Node.js

- Open source runtime environment
 - Develop server side web applications
 - Act as a stand alone web server



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



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
- rootJS bindings realized as native Node.js module written in C++



- Task: encapsulation of ROOT objects and functions
 - Scanning ROOT structures during initialization
 - Encapsulating objects with heavily nested object structures
 - Introduce (proxy) object cache

- Task: encapsulation of ROOT objects and functions
 - Scanning ROOT structures during initialization
 - Encapsulating objects with heavily nested object structures
 - Introduce (proxy) object cache
- ⇒ Generally negligible hardware requirements of the bindings themselves

Product data

The following data will be stored by the rootJS bindings

- All ROOT classes and methods as they dynamically mapped to their JavaScript equivalents
- ROOT environment state
- Application context is derived from TApplication
- Map of v8::handles 2 identified by the address of ROOT objects

Scenarios

rootJS is used by applications to access the ROOT framework

Scenarios

rootJS is used by applications to access the ROOT framework
⇒ our users are those applications

Event Viewer

Event Viewer

Event Viewers provide visualisation of experimental data

Event Viewer

Event Viewers provide visualisation of experimental data

- useful for quick eyescan of data

Event Viewer

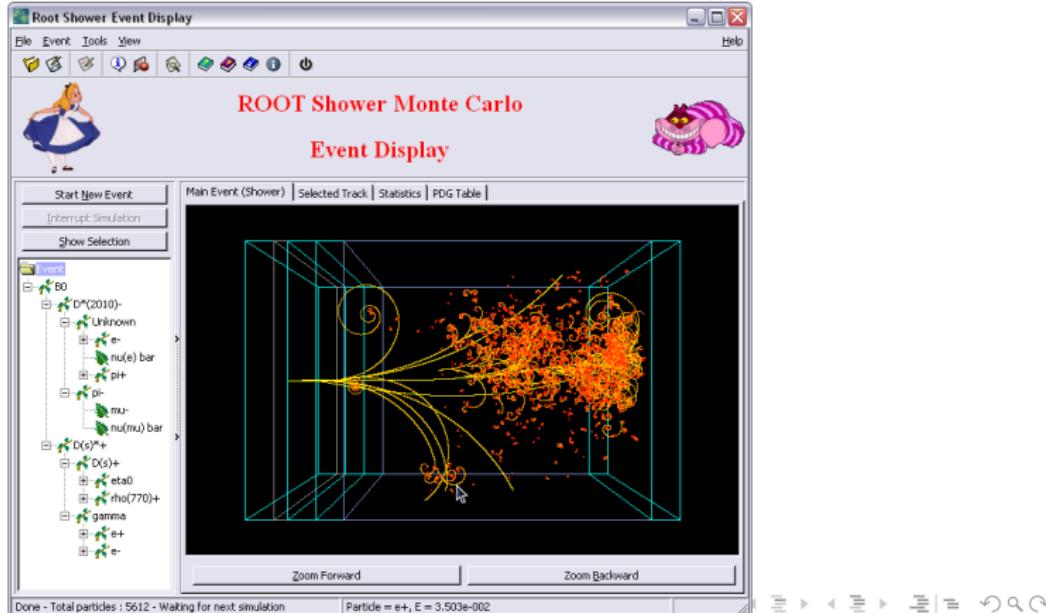
Event Viewers provide visualisation of experimental data

- useful for quick eyescan of data
- check if data is recorded properly

Event Viewer

Event Viewers provide visualisation of experimental data

- useful for quick eyescan of data
- check if data is recorded properly



PSE	Purpose	Usage	Environment	Data	Scenarios	Use Cases	System Model	Test Cases
C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart	– rootjs	oo	ooo	oooo	●oooo	oooooooo	ooooo	15/33

Event Viewer

Conventional ROOT Event Viewer

Event Viewer

Conventional ROOT Event Viewer

- standalone ROOT application

Event Viewer

Conventional ROOT Event Viewer

- standalone ROOT application
- requires ROOT on machine

Event Viewer

Conventional ROOT Event Viewer

- standalone ROOT application
- requires ROOT on machine
- requires ROOT's dependencies

Event Viewer

Conventional ROOT Event Viewer

- standalone ROOT application
- requires ROOT on machine
- requires ROOT's dependencies
- requires access to data source

Event Viewer

Conventional ROOT Event Viewer

- standalone ROOT application
- requires ROOT on machine
- requires ROOT's dependencies
- requires access to data source

⇒ very limited portability and harsh requirements for client system

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

- server runs ROOT and its dependencies

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

- server runs ROOT and its dependencies
- no access to critical data sources required

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

- server runs ROOT and its dependencies
- no access to critical data sources required
- no heavy work load on client system

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

- server runs ROOT and its dependencies
- no access to critical data sources required
- no heavy work load on client system
- client only needs modern web browser

Event Viewer

Client/Server based Web Event Viewer using rootJS and nodeJS

- server runs ROOT and its dependencies
- no access to critical data sources required
- no heavy work load on client system
- client only needs modern web browser

⇒ great portability and ease of use as client can be almost any device

Event Viewer

<i>Scenario name</i>	EventViewer
<i>Participating actors</i>	<u>Server:EventViewerServer; :ROOT</u> <u>Client:EventViewerClient; :rootJS</u>
<i>Flow of events</i>	

- Client requests updates from Server.
- Server interfaces with ROOT through rootJS.
- ROOT's I/O accesses and processes data
- ROOT returns the data to the Server using rootJS.
- Server sends the data to the Client
- Client renders data locally.

Scenarios

In what ways could these bindings also improve work efficiency for scientists?

Scenarios

In what ways could these bindings also improve work efficiency for scientists?

- integrating run logs and quality assurance in ROOT workflow

Scenarios

In what ways could these bindings also improve work efficiency for scientists?

- integrating run logs and quality assurance in ROOT workflow
- ...

UseROOTGlobal

<i>Use case name</i>	UseROOTGlobal
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS; Communicates with ROOT

Flow of events

UseROOTGlobal

<i>Use case name</i>	UseROOTGlobal
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS; Communicates with ROOT

Flow of events

- ➊ The NodeJSApplication requests access to a global variable of ROOT.

UseROOTGlobal

<i>Use case name</i>	UseROOTGlobal
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS; Communicates with ROOT

Flow of events

- ① The NodeJSApplication requests access to a global variable of ROOT.
- ② rootJS sends a request to the corresponding ROOT variable.

UseROOTGlobal

<i>Use case name</i>	UseROOTGlobal
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS; Communicates with ROOT

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.

UseROOTGlobal

<i>Use case name</i>	UseROOTGlobal
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS; Communicates with ROOT

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.

UseROOTGlobal

Entry condition

rootJS has been initialized.

Exit condition

The value has been returned to the NodeJSApplication.

UseROOTObject

<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>	

UseROOTObject

<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.

UseROOTObject

<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.

UseROOTObject

Flow of events

UseROOTObject

Flow of events

- ③ rootJS stores the created ProxyObject in a cache memory.

UseROOTObject

Flow of events

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

Flow of events

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

rootJS has been initialized.

Entry condition

Exit condition

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

UseROOTFunction

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

UseROOTFunction

<i>Use case name</i>	UseROOTFunction
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, ProxyObject; Communicates with ROOT

Flow of events

- 1 The NodeJSApplication requests access to a ROOT function.

UseROOTFunction

<i>Use case name</i>	UseROOTFunction
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, ProxyObject; Communicates with ROOT

Flow of events

- ① The NodeJSApplication requests access to a ROOT function.
- ② rootJS calls the corresponding ROOT function.

UseROOTFunction

<i>Use case name</i>	UseROOTFunction
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, ProxyObject; Communicates with ROOT

Flow of events

- ① The NodeJSApplication requests access to a ROOT function.
- ② rootJS calls the corresponding ROOT function.
- ③ ROOT responds.

UseROOTFunction

Flow of events

UseROOTFunction

Flow of events

- ④ rootJS encapsulates the returned ROOT object within a ProxyObject.

UseROOTFunction

Flow of events

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

UseROOTFunction

Flow of events

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

rootJS has been initialized.

Entry condition

Exit condition

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

<i>Use Case name</i>	UseJIT
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, Cling; Communicates with ROOT
<i>Flow of events</i>	

<i>Use Case name</i>	UseJIT
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, Cling; Communicates with ROOT

Flow of events

- 1 The NodeJSApplication wants to execute ROOT specific C++ code (given as string) during runtime.

<i>Use Case name</i>	UseJIT
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, Cling; Communicates with ROOT

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.

<i>Use Case name</i>	UseJIT
<i>Participating actor instances</i>	Initiated by NodeJSApplication; Processed by rootJS, Cling; Communicates with ROOT
<i>Flow of events</i>	

- 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

Flow of events

- ③ rootJS takes care of encapsulating exceptions possibly thrown by Cling or ROOT during evaluation and execution.

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.
-

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.

rootJS and Cling have been initialized.

Entry condition

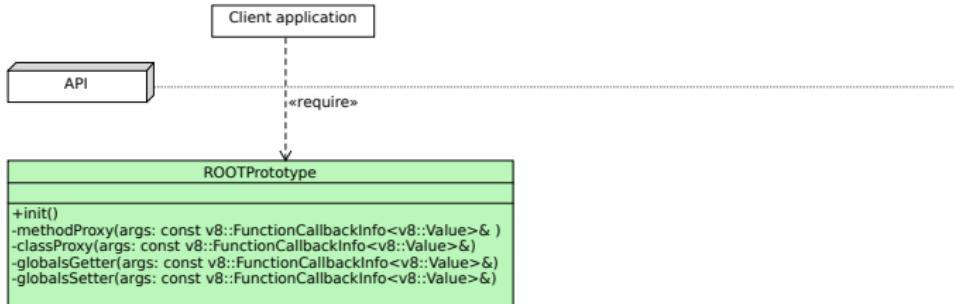
Exit condition rootJS either confirms the proper execution of the specified instructions or forwards thrown exceptions to the NodeJSApplication.

Basic Architecture

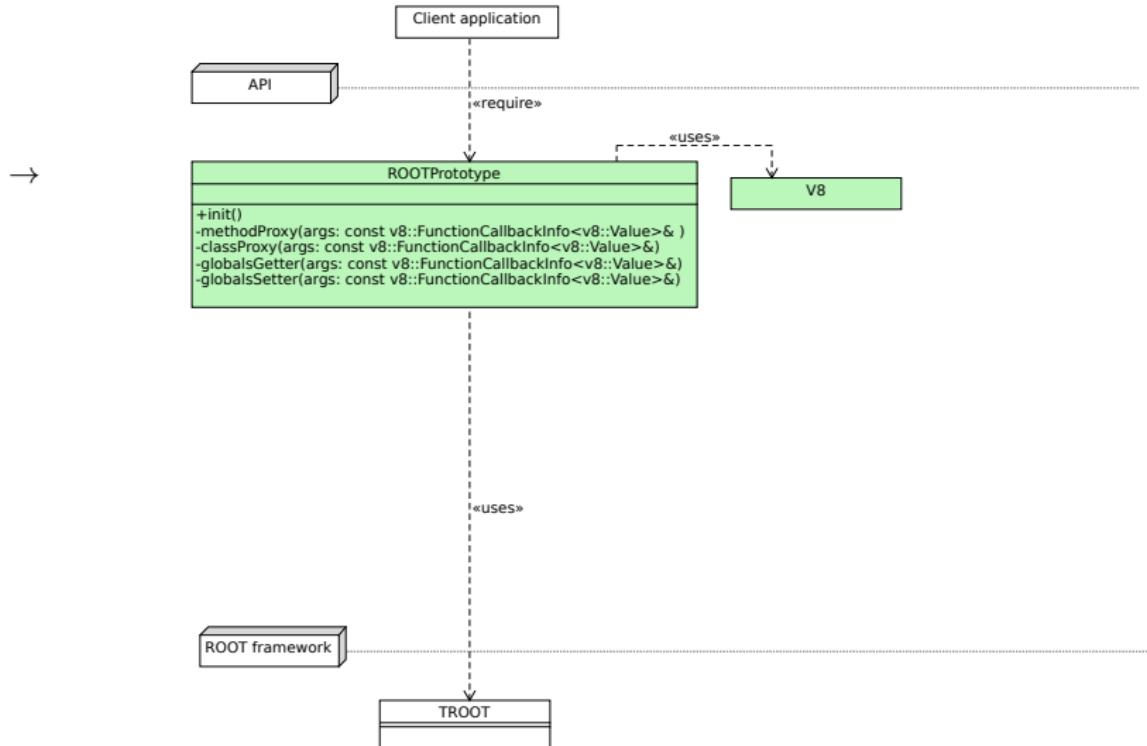


Basic Architecture

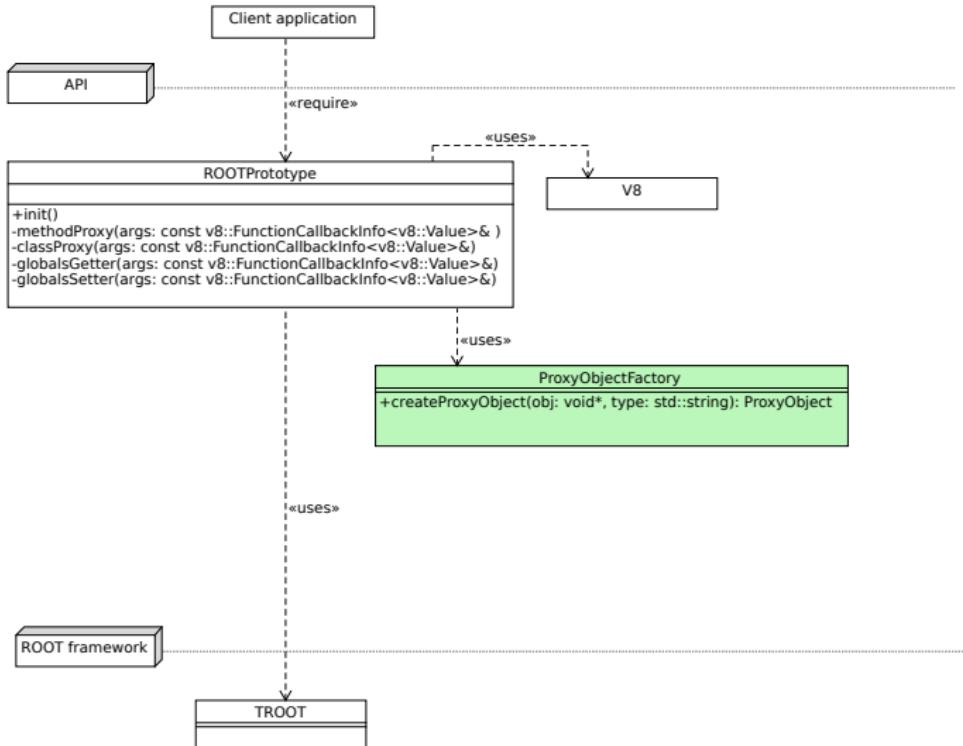
→



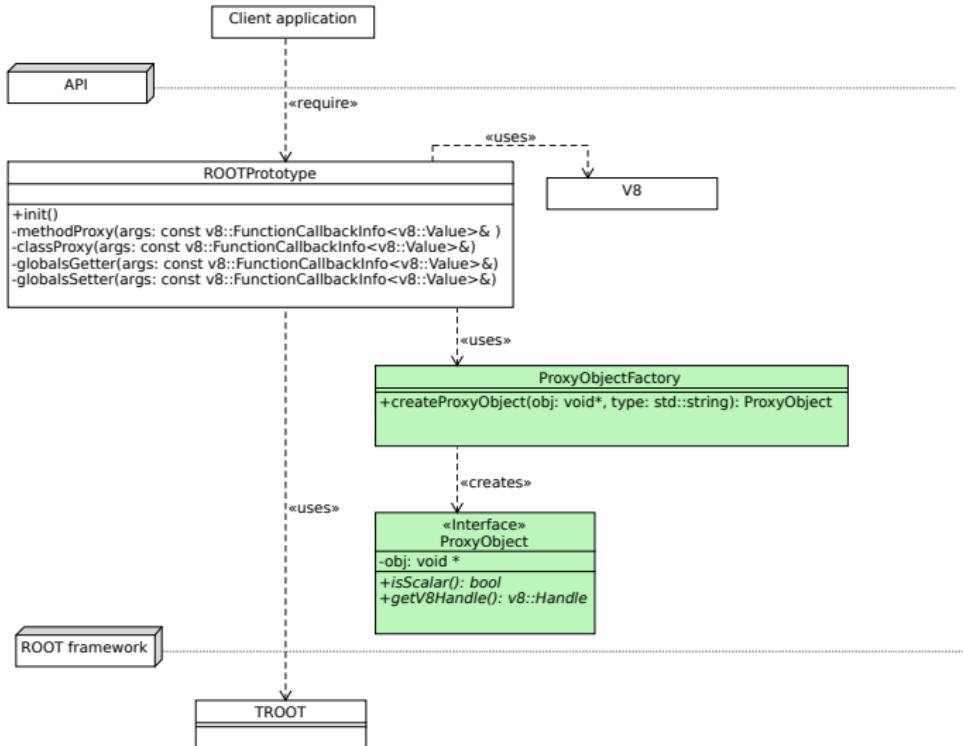
Basic Architecture



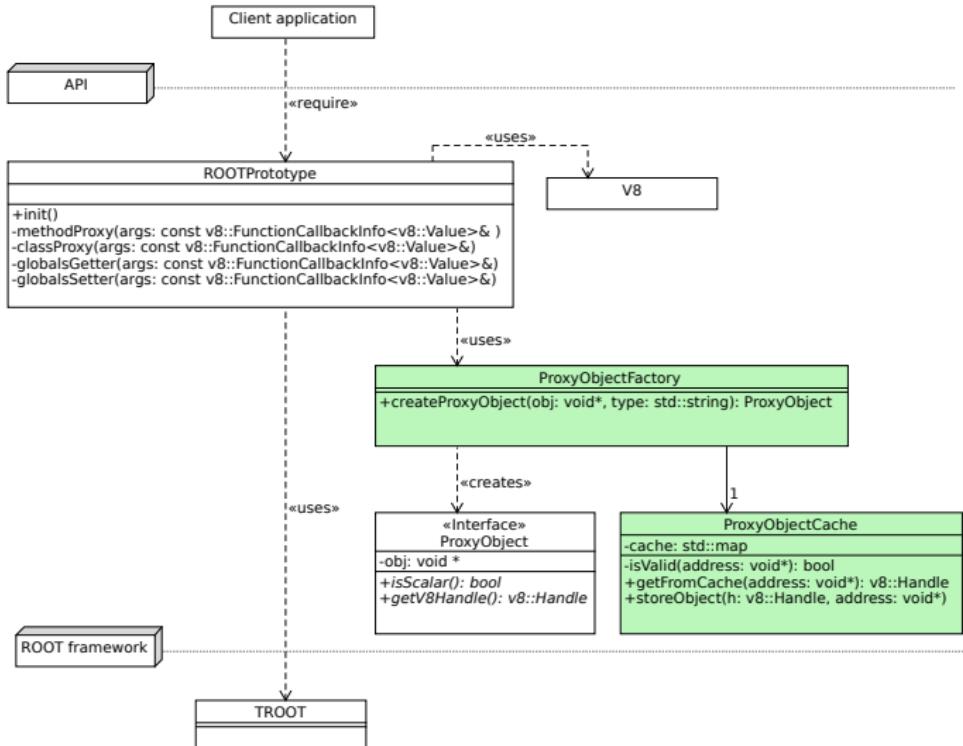
Basic Architecture



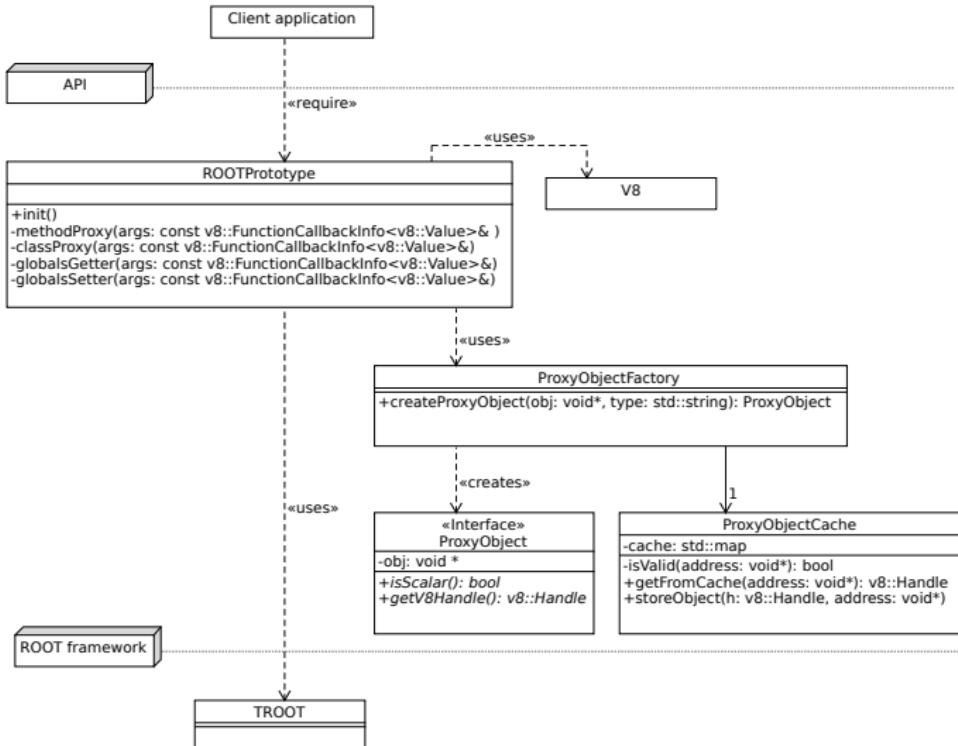
Basic Architecture



Basic Architecture



Basic Architecture



Initialization

- Expose all
 - Global variables
 - Global functions
 - Classes

Initialization

- 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

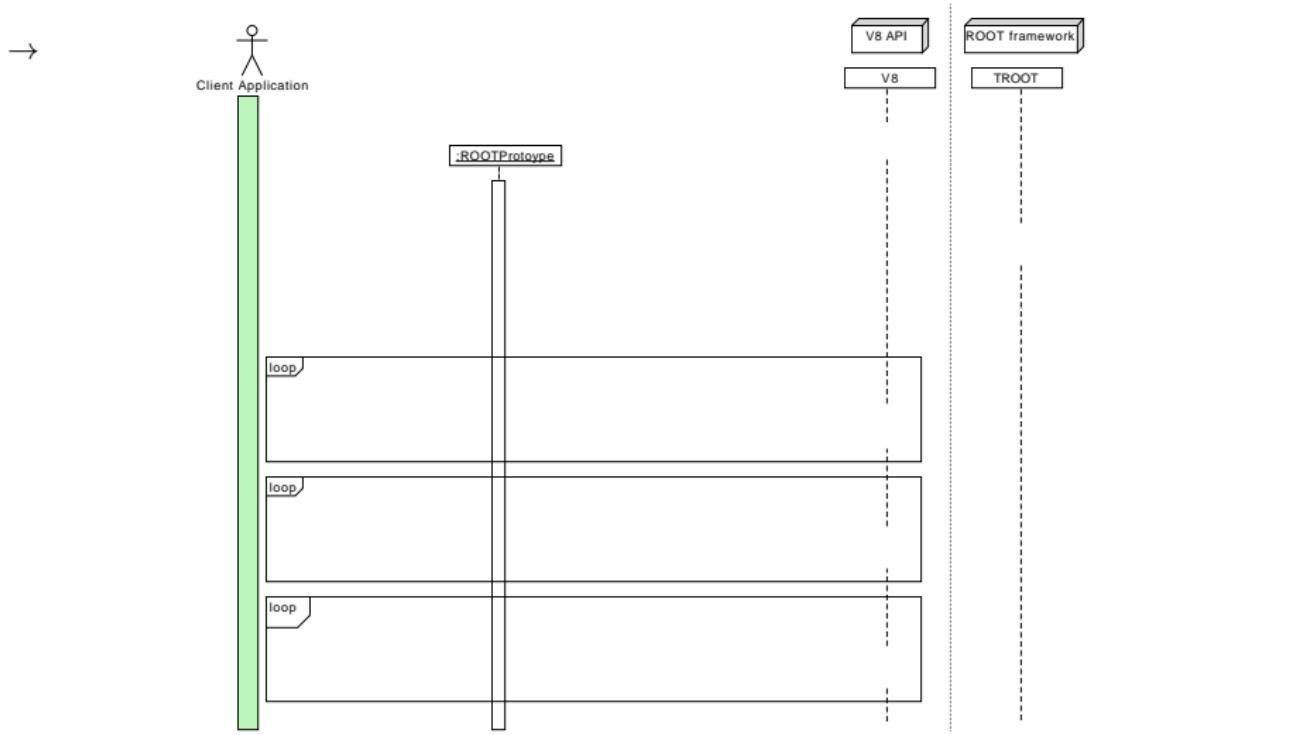
Initialization

- 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

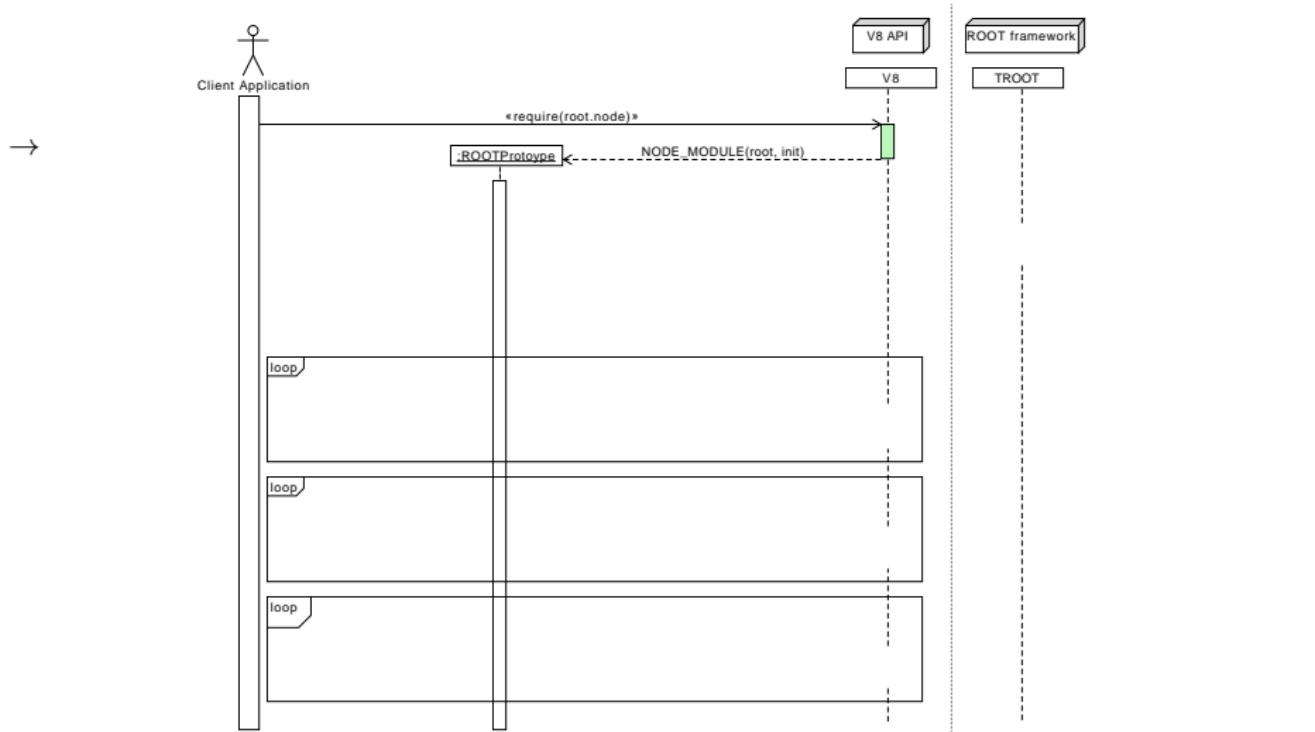
Names

- Functions and classes have the same name as in Root
- Global variables can be called using Get[Variable] and Set[Variable] methods

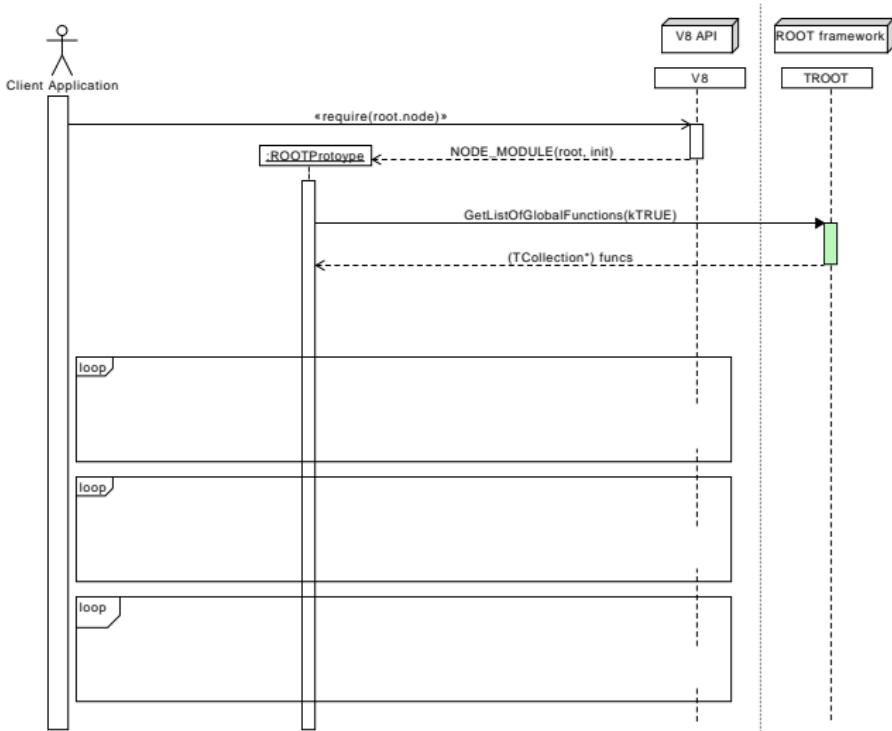
rootJS init



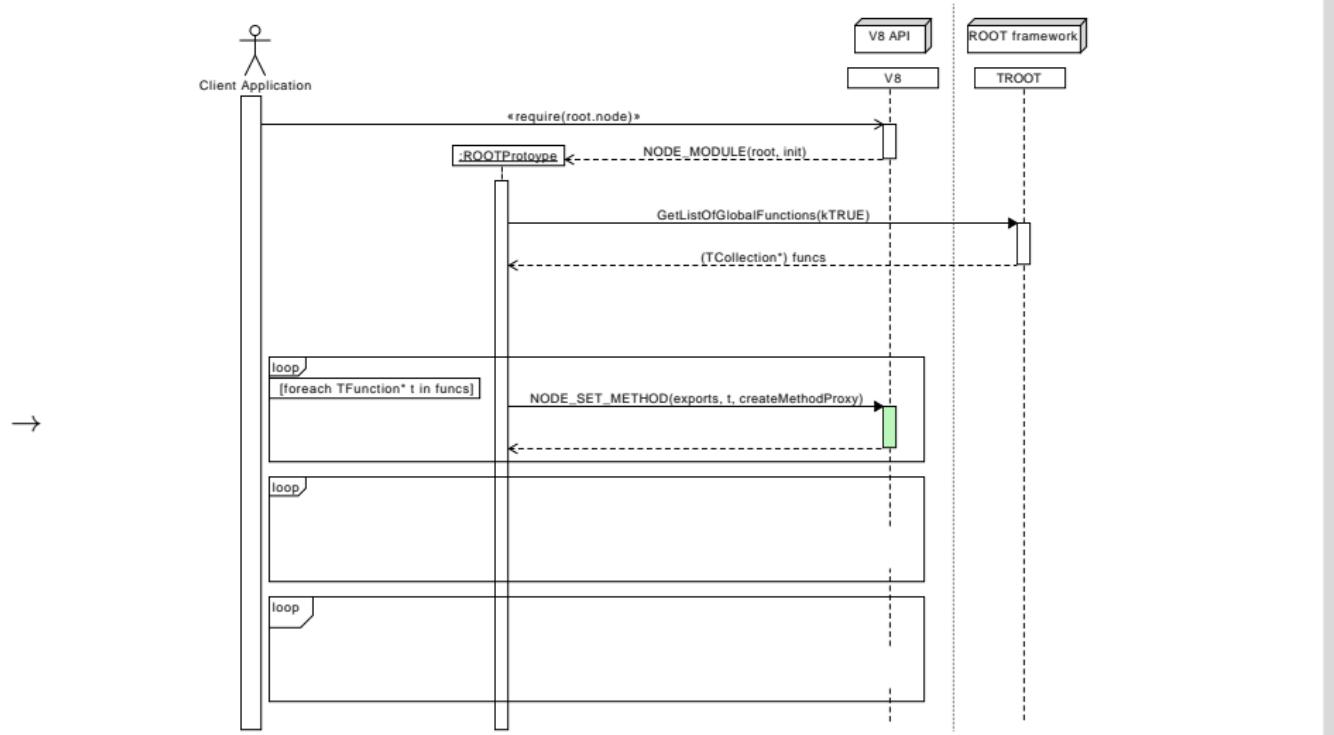
rootJS init



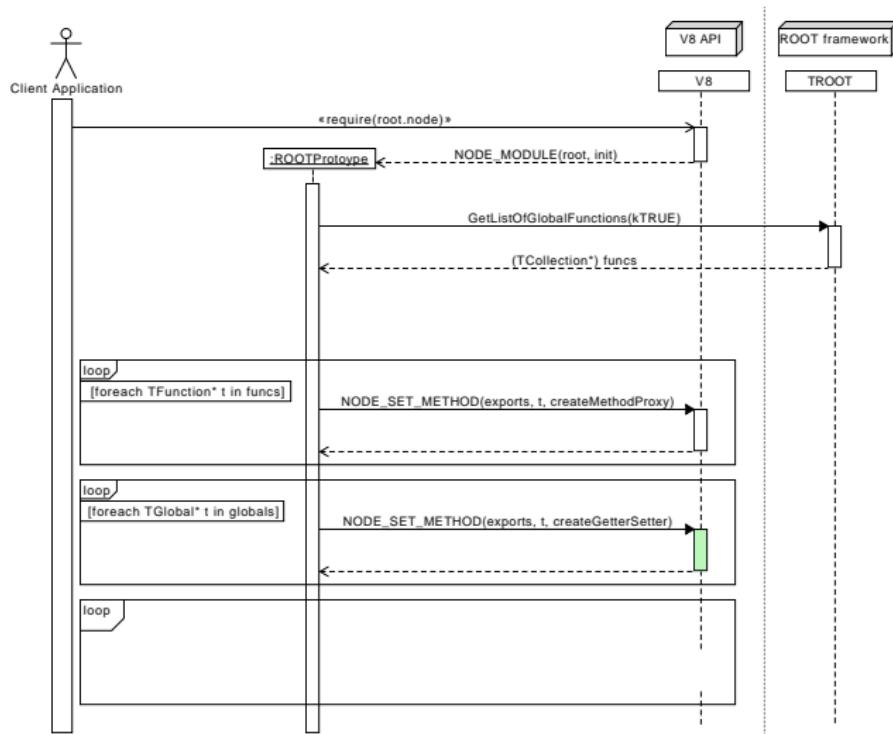
rootJS init



rootJS init



rootJS init



Call a feature

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

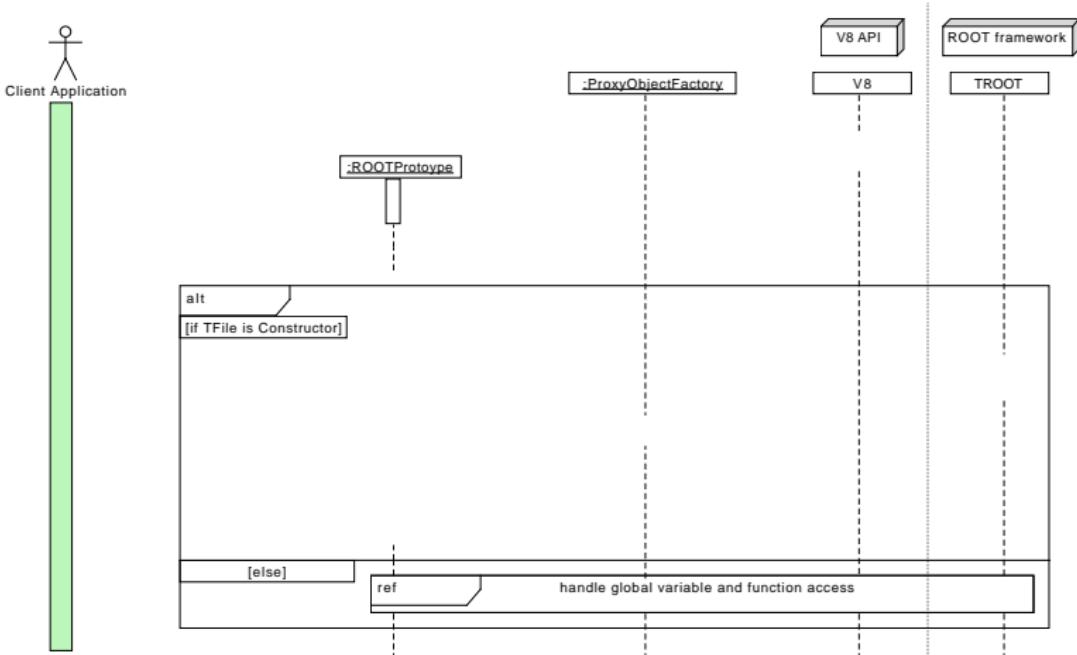
Call a feature

- 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

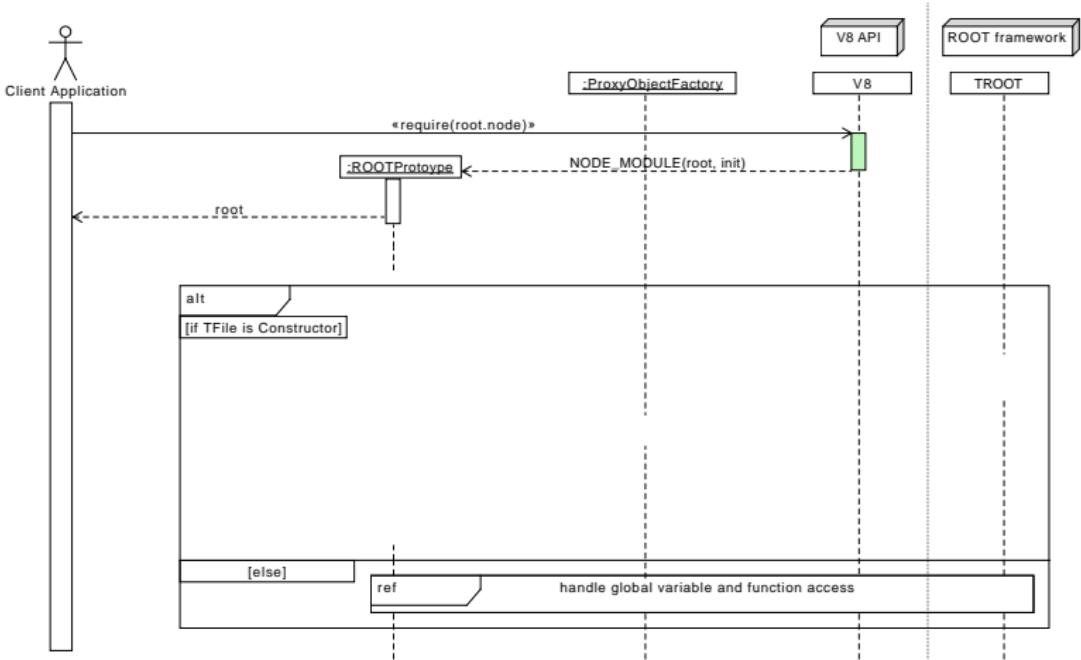
Call a feature

- 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

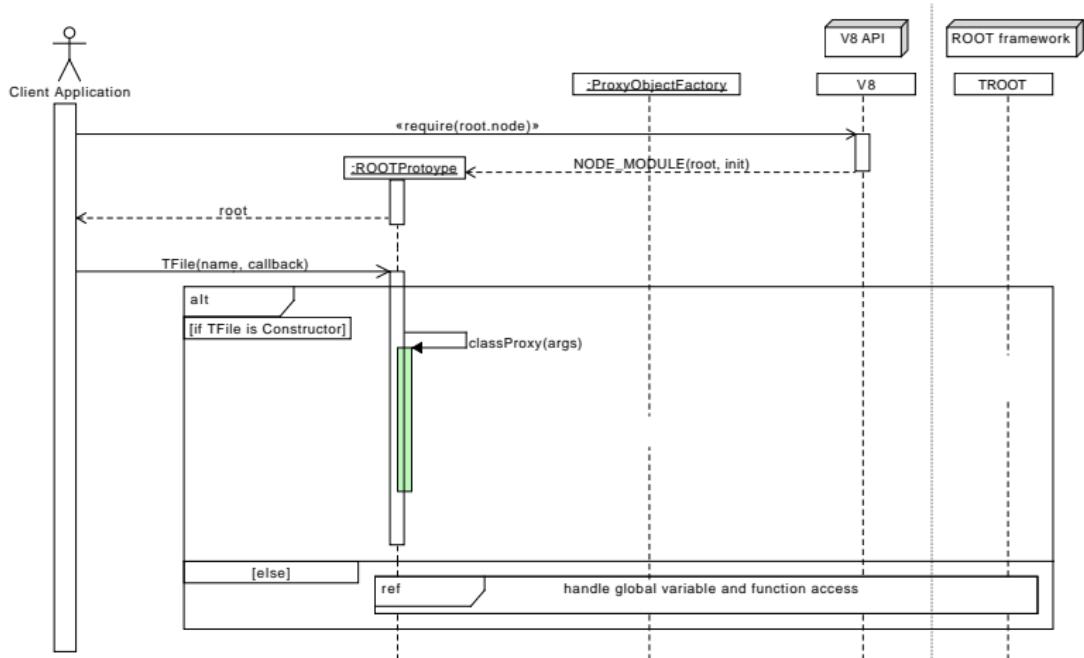
proxied file access



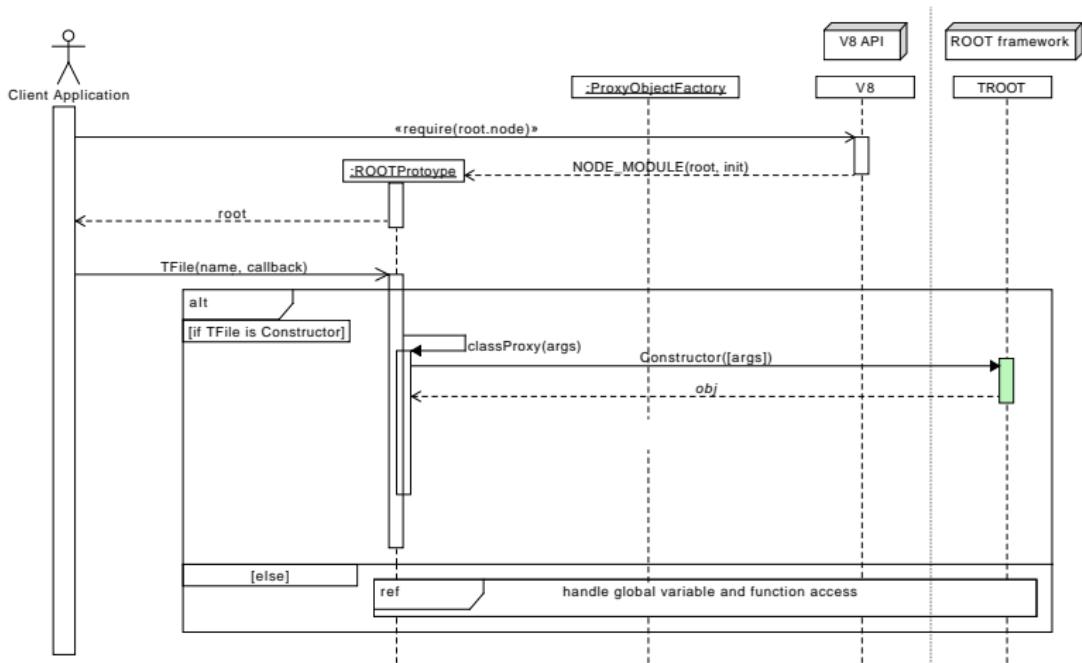
proxied file access



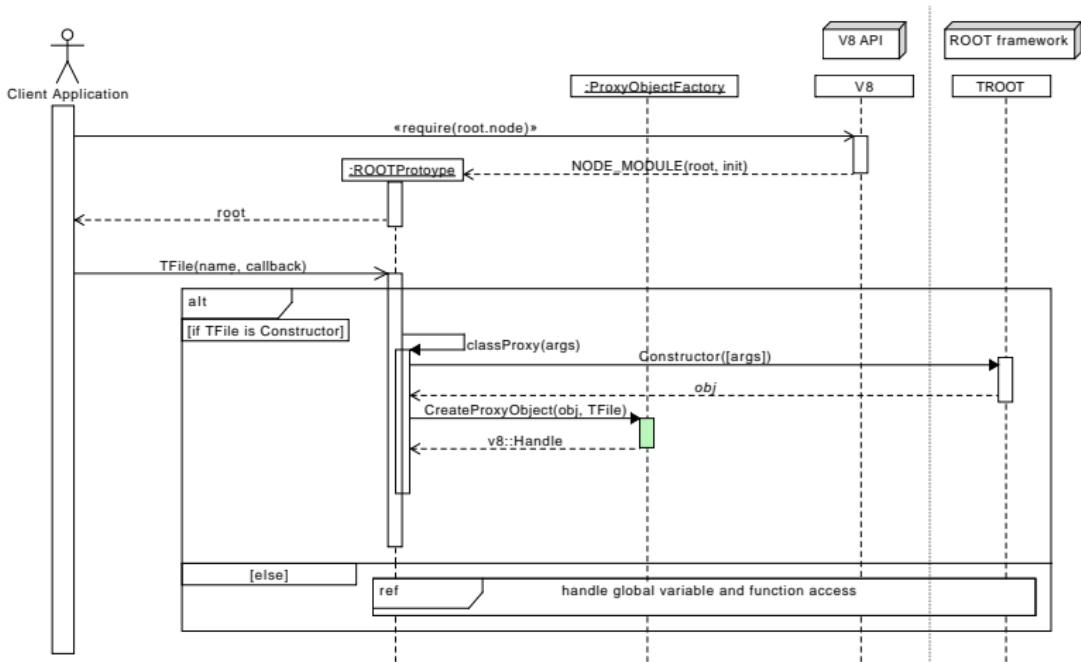
proxied file access



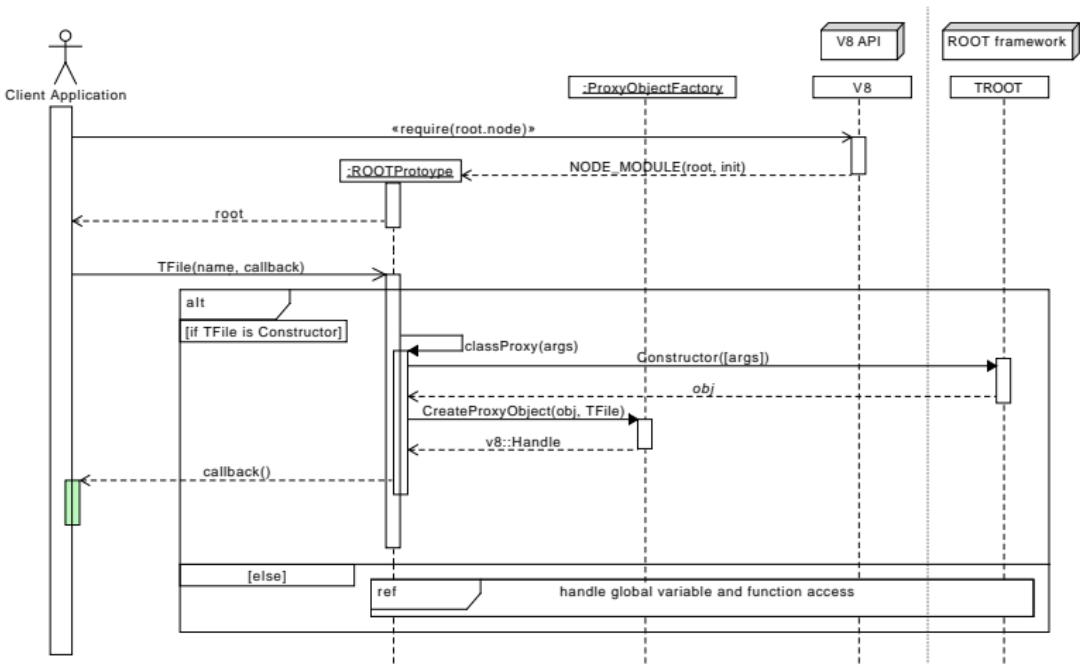
proxied file access



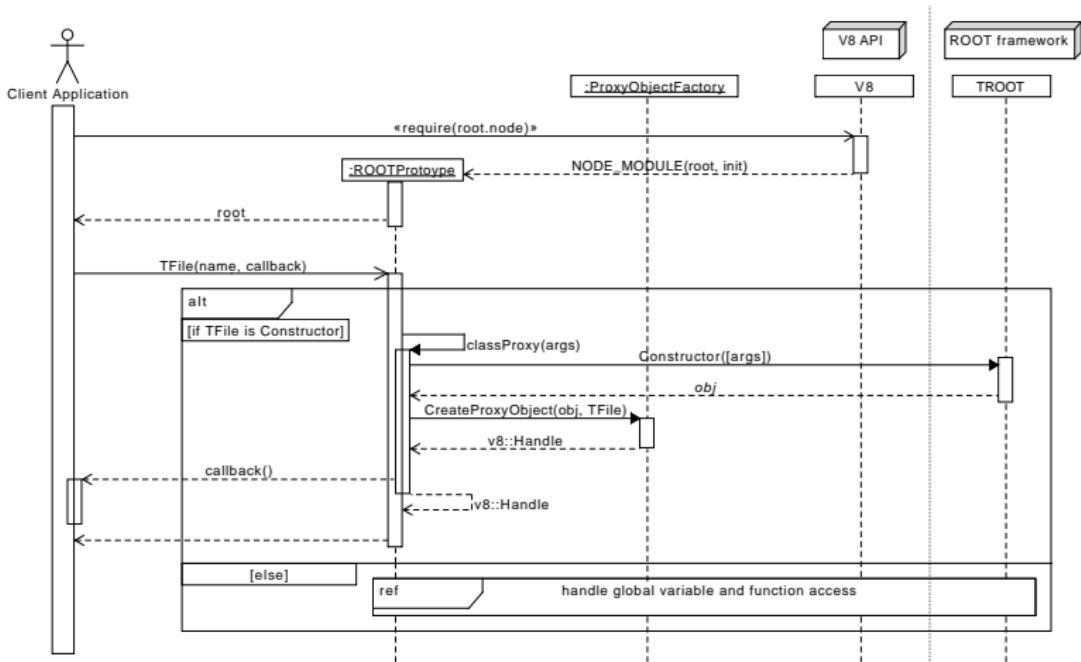
proxied file access



proxied file access



proxied file access



Test Cases

- 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

References I

-  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.

References II

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