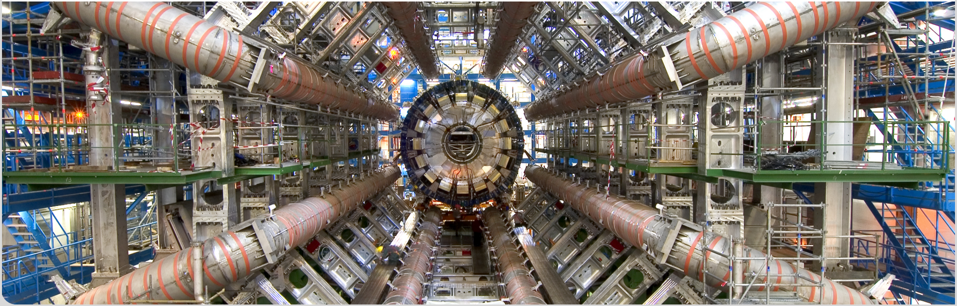
rootJS - Specification

PSE - Software Engineering Practice

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

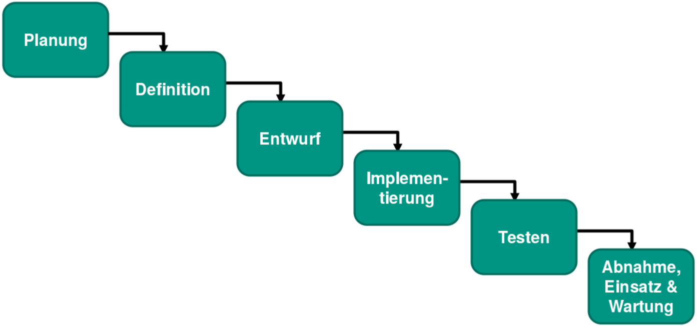
STEINBUCH CENTER FOR COMPUTING



About PSE

Praxis der Softwareentwicklung(PSE) = Software Engineering Practice

- Waterfall model
  - Planning/definition



Planung

Definition

Entwurf

Implementierung

Testen

Abnahme, Einsatz & Wartung

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

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

Audience

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++.

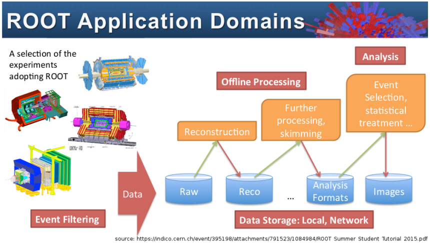
- Scientists (e.g. particle physicists)
- Researchers
- Web-developers interested in creating applications based on ROOT

Operating conditions

- rootJS will be used on servers that run ROOT and have access to the required data sources.
- As ROOT 6 currently runs on Linux and OS X only, usage of the bindings is limited to those platforms.

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



ROOT Application Domains

A selection of the experiments adopting ROOT

Analysis

Offline Processing

Event Selection, statistical treatment ...

Further processing, skimming

Reconstruction

Data

Raw

Reco

...

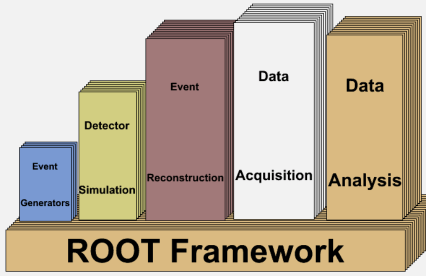
Analysis Formats

Images

Event Filtering

Data Storage: Local, Network

source: https://indico.cern.ch/event/395198/attachments/791523/1084984/ROOT Summer Student Tutorial 2015.pdf



Event Generators

Detector Simulation

Event Reconstruction

Data Acquisition

Data Analysis

ROOT Framework

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++

Hardware

- Task: encapsulation of ROOT objects and functions
  - → scanning ROOT structures during initialization
  - → encapsulating objects with heavily nested object structures
  - → introduce (proxy) object cache

Hardware

- 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

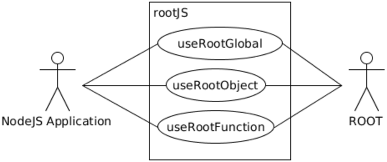
Product data

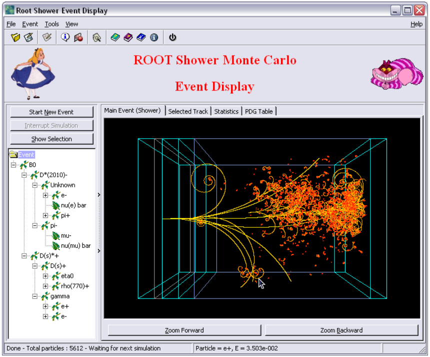
The following data will be stored by the rootJS bindings

- 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

Product interface

Event Viewer

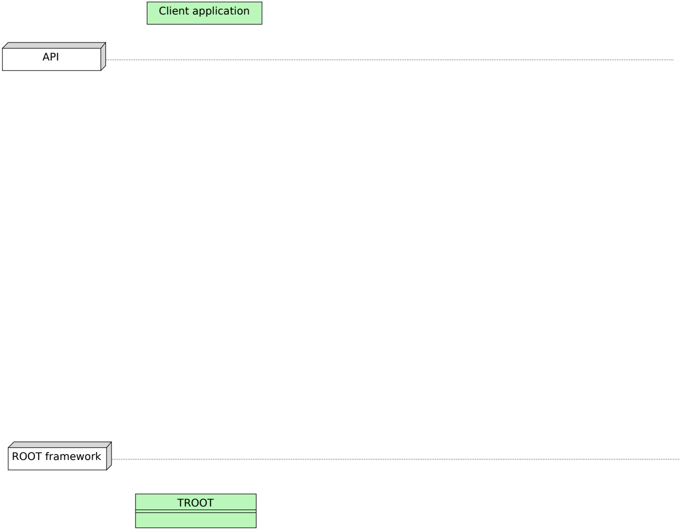




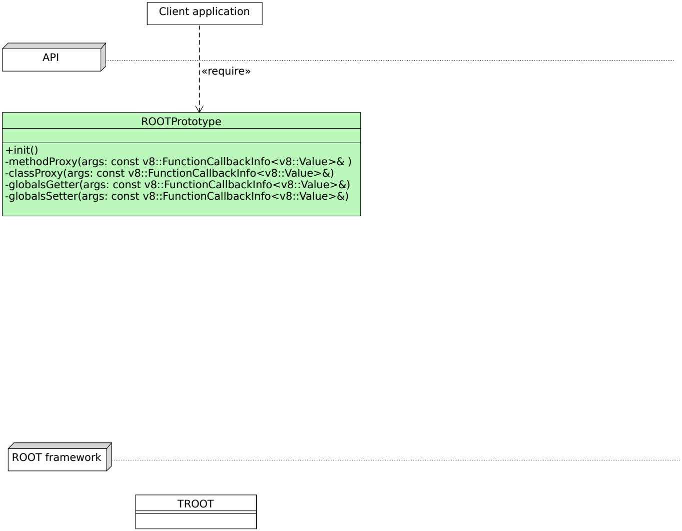
Done - Total particles : 5612 - Waiting for next simulation

Particle = e+, E = 3.503e-002

Basic Architecture



Basic Architecture



Client application

API

«require»

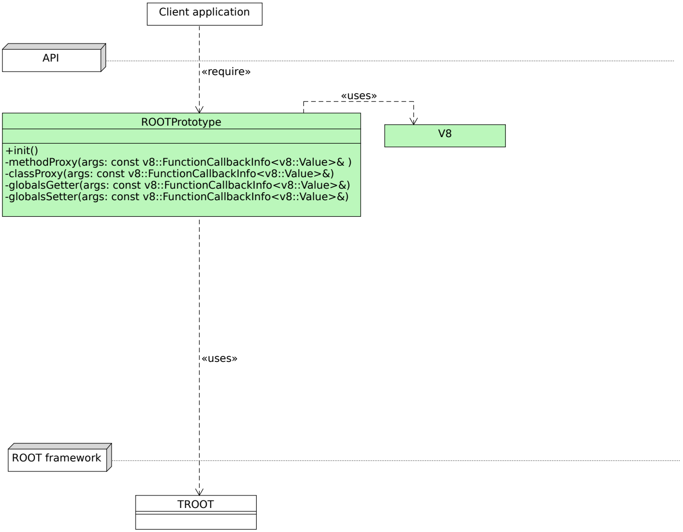
ROOTPrototype

+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)

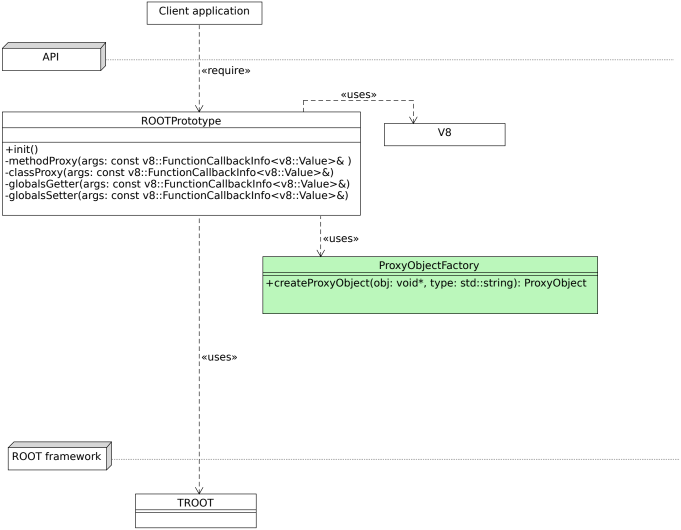
ROOT framework

TROOT

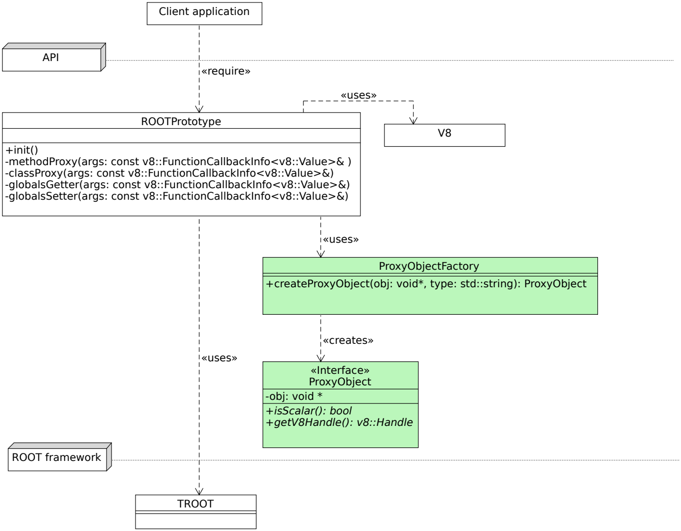
Basic Architecture


Client application
API
«require»
«uses»
ROOTPrototype
+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
V8
«uses»
ROOT framework
TROOT

Basic Architecture


Client application
API
«require»
«uses»
V8
ROOTPrototype
+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
«uses»
ProxyObjectFactory
+createProxyObject(obj: void*, type: std::string): ProxyObject
«uses»
ROOT framework
TROOT

Basic Architecture



Client application

API

«require»

«uses»

ROOTPrototype

+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)

V8

«uses»

ProxyObjectFactory

+createProxyObject(obj: void*, type: std::string): ProxyObject

«creates»

«uses»

«Interface»
ProxyObject

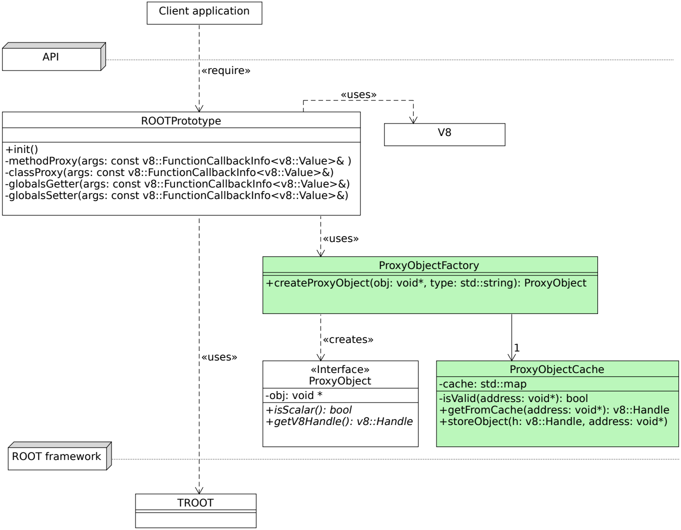
-obj: void *

+isScalar(): bool
+getV8Handle(): v8::Handle

ROOT framework

TROOT

Basic Architecture



Client application

API

«require»

«uses»

ROOTPrototype

+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)

V8

«uses»

ProxyObjectFactory

+createProxyObject(obj: void*, type: std::string): ProxyObject

«uses»

«creates»

1

«Interface»
ProxyObject

-obj: void *

+isScalar(): bool
+getV8Handle(): v8::Handle

ProxyObjectCache

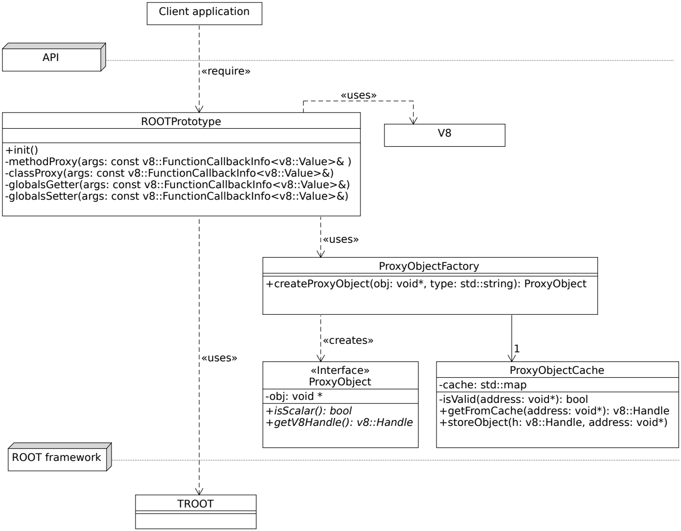
-cache: std::map

-isValid(address: void*): bool
+getFromCache(address: void*): v8::Handle
+storeObject(h: v8::Handle, address: void*)

ROOT framework

TROOT

Basic Architecture


Client application
API
«require»
«uses»
V8
ROOTPrototype
+init()
-methodProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-classProxy(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsGetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
-globalsSetter(args: const v8::FunctionCallbackInfo<v8::Value>&)
«uses»
ProxyObjectFactory
+createProxyObject(obj: void*, type: std::string): ProxyObject
«creates»
1
«uses»
«Interface»
ProxyObject
-obj: void *
+isScalar(): bool
+getV8Handle(): v8::Handle
ProxyObjectCache
-cache: std::map
-isValid(address: void*): bool
+getFromCache(address: void*): v8::Handle
+storeObject(h: v8::Handle, address: void*)
ROOT framework
TROOT

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 beeing passed to node

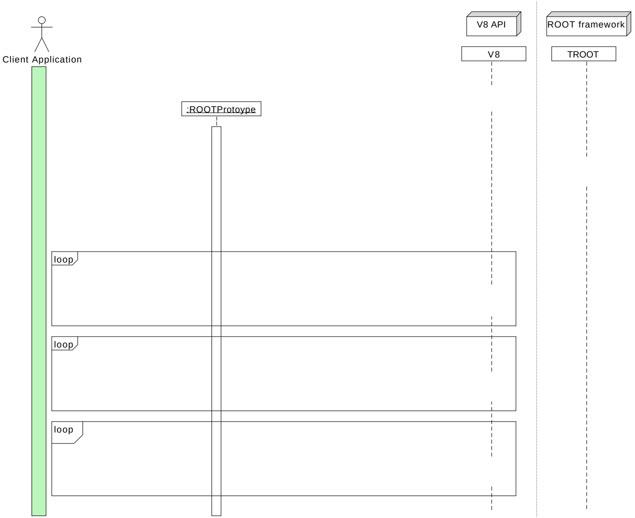
Initialization

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

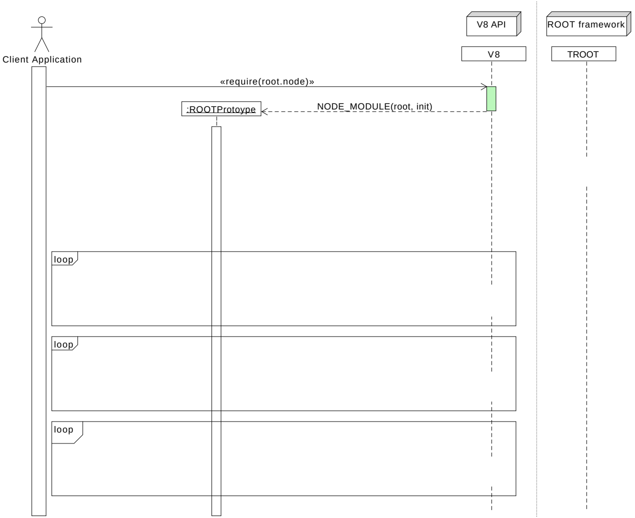
Names

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

rootJS init

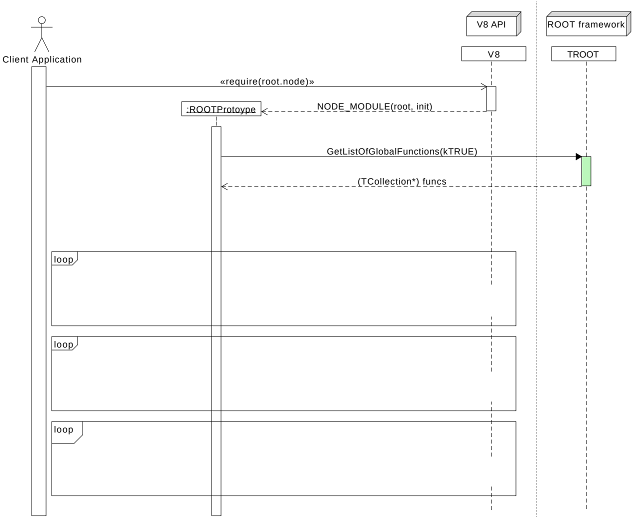

Client Application
:ROOTProtoype
V8 API
V8
ROOT framework
TROOT
loop
loop
loop

rootJS init



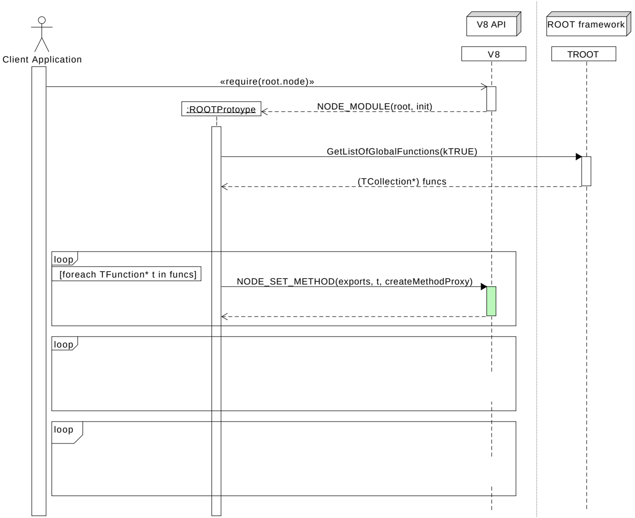
Client Application
V8 API
ROOT framework
V8
TROOT
«require(root.node)»
:ROOTProtoype
NODE_MODULE(root, init)
loop
loop
loop

rootJS init

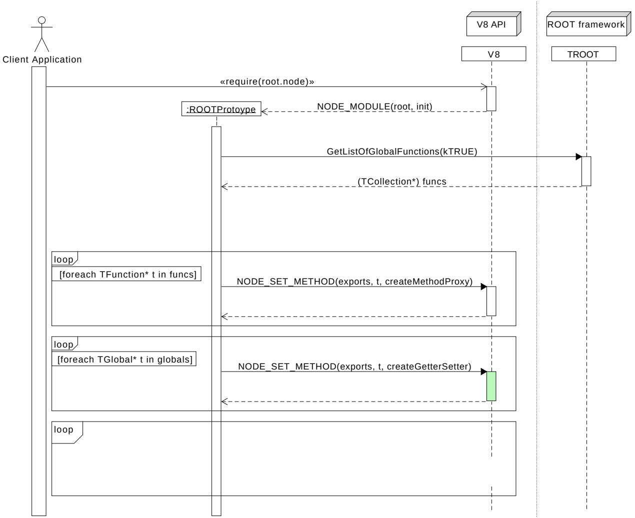


Client Application
V8 API
ROOT framework
V8
TROOT
«require(root.node)»
:ROOTProtoype
NODE_MODULE(root, init)
GetListOfGlobalFunctions(kTRUE)
(TCollection*) funcs
loop
loop
loop

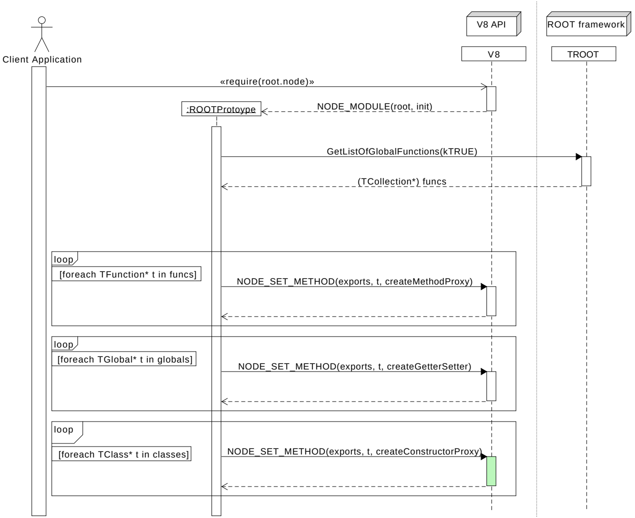
rootJS init


V8 API
ROOT framework
Client Application
V8
TROOT
«require(root.node)»
:ROOTProtoype
NODE_MODULE(root, init)
GetListOfGlobalFunctions(kTRUE)
(TCollection*) funcs
loop
[foreach TFunction* t in funcs]
NODE_SET_METHOD(exports, t, createMethodProxy)
loop
loop

rootJS init

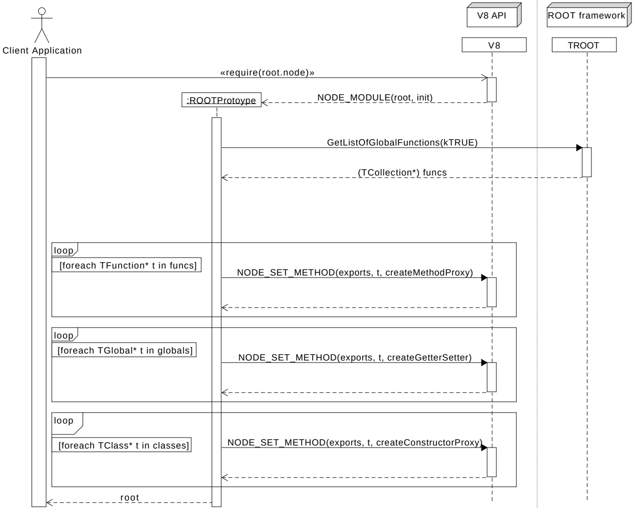

Client Application
V8 API
ROOT framework
V8
TROOT
«require(root.node)»
:ROOTProtoype
NODE_MODULE(root, init)
GetListOfGlobalFunctions(kTRUE)
(TCollection*) funcs
loop
[foreach TFunction* t in funcs]
NODE_SET_METHOD(exports, t, createMethodProxy)
loop
[foreach TGlobal* t in globals]
NODE_SET_METHOD(exports, t, createGetterSetter)
loop

rootJS init



V8 API
ROOT framework
Client Application
V8
TROOT
«require(root.node)»
:ROOTProtoype
NODE_MODULE(root, init)
GetListOfGlobalFunctions(kTRUE)
(TCollection*) funcs
loop
[foreach TFunction* t in funcs]
NODE_SET_METHOD(exports, t, createMethodProxy)
loop
[foreach TGlobal* t in globals]
NODE_SET_METHOD(exports, t, createGetterSetter)
loop
[foreach TClass* t in classes]
NODE_SET_METHOD(exports, t, createConstructorProxy)

rootJS init



Client Application

V8 API

V8

ROOT framework

TROOT

:ROOTProtoype

«require(root.node)»

NODE_MODULE(root, init)

GetListOfGlobalFunctions(kTRUE)

(TCollection*) funcs

loop

[foreach TFunction* t in funcs]

NODE_SET_METHOD(exports, t, createMethodProxy)

loop

[foreach TGlobal* t in globals]

NODE_SET_METHOD(exports, t, createGetterSetter)

loop

[foreach TClass* t in classes]

NODE_SET_METHOD(exports, t, createConstructorProxy)

root

Call a feature

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

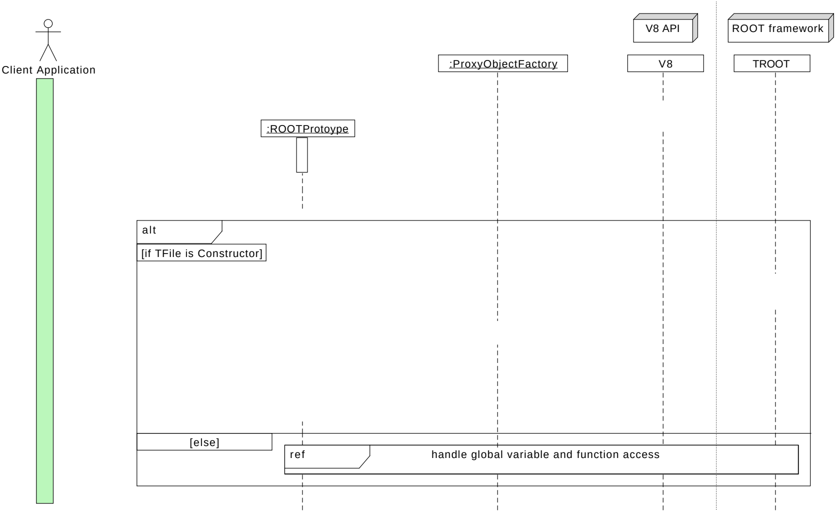
Call a feature

- 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

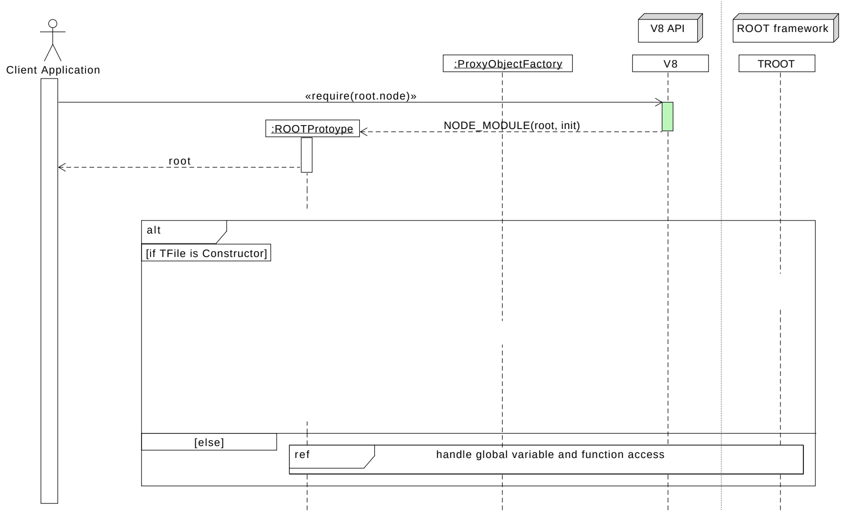
Call a feature

- 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

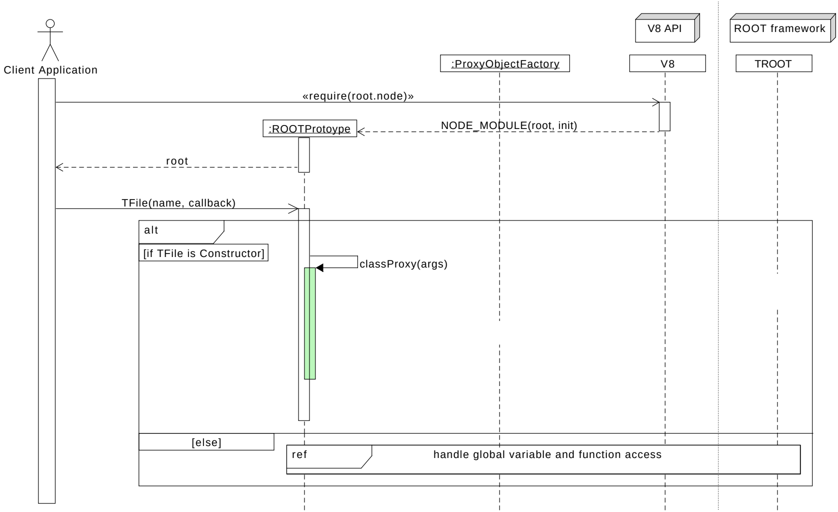
proxied file access



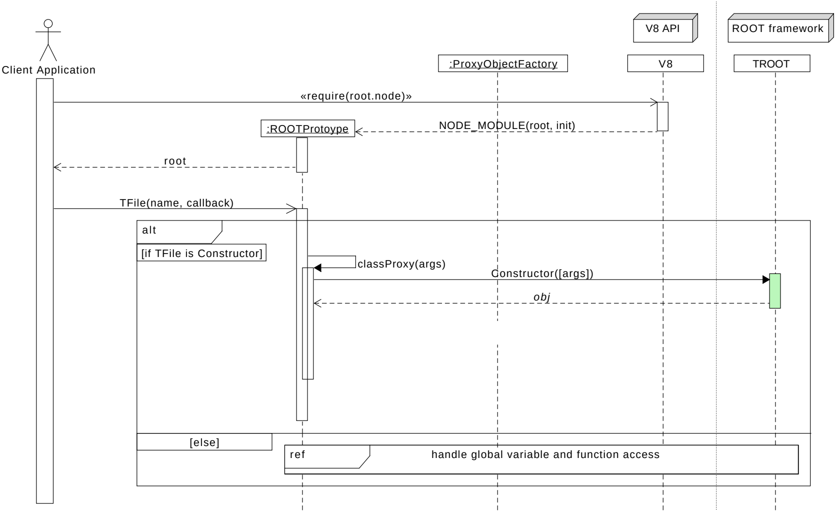
proxied file access



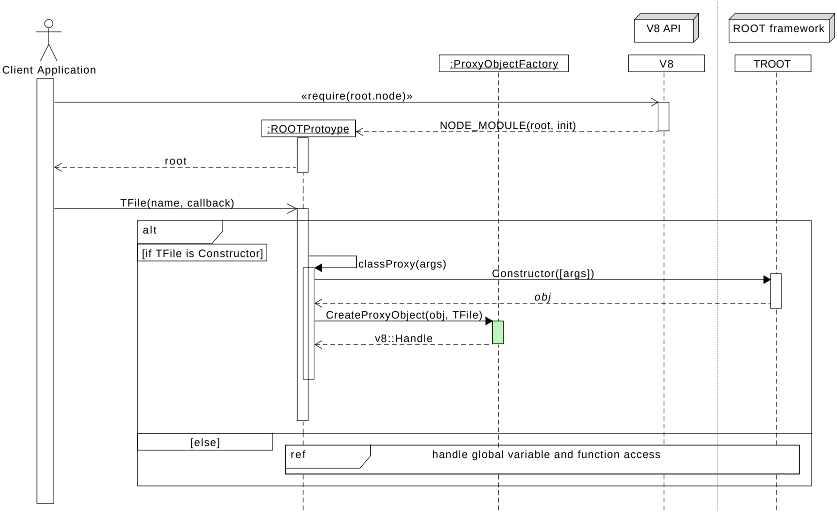
proxied file access



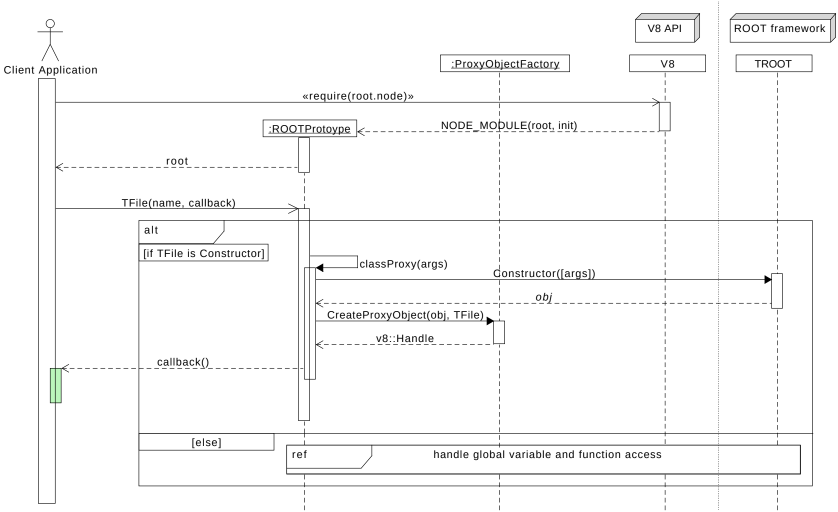
proxied file access



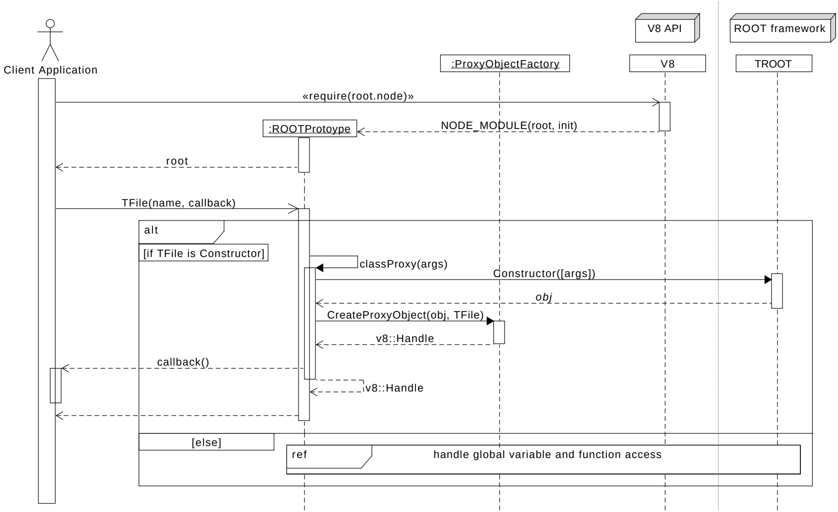
proxied file access



proxied file access

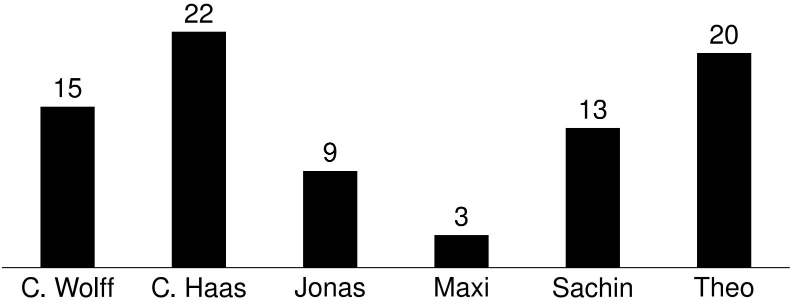


proxied file access

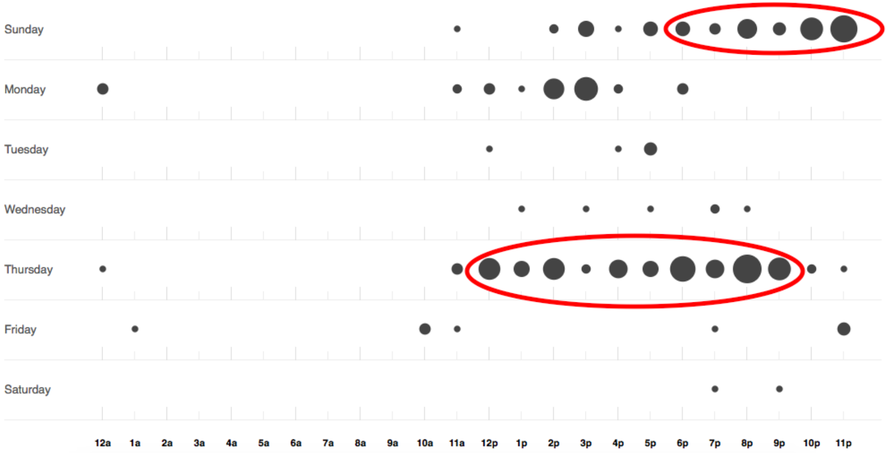


Test Cases

Merges



Punchcard


Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
12a
1a
2a
3a
4a
5a
6a
7a
8a
9a
10a
11a
12p
1p
2p
3p
4p
5p
6p
7p
8p
9p
10p
11p

References I