



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

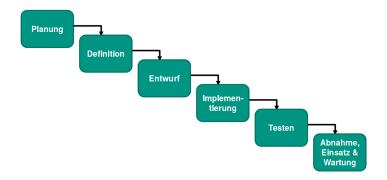


About PSE



Praxis der Softwareentwicklung(PSE) = Software Engineering Practice

- Waterfall model
 - Planning/definition



Environment Data Interface Scenarios Use Cases System Model



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

Interface

- update dynamically following changes to C++ internals
- provide asynchronous wrappers for common I/O operations (i.e. file and tree access)

Scenarios Use Cases

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)

Environment Data Interface Scenarios Use Cases

Limiting criteria



The bindings should not

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

Environment Data Interface Scenarios Use Cases

December 15, 2015

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)

Scenarios Use Cases

Run on any platform that supports a browser

Interface



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

Data Interface Scenarios Use Cases

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.

Scenarios Use Cases

Interface

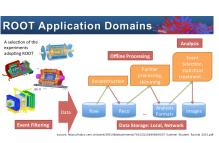
ROOT



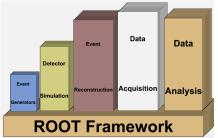
- process and visualize large amounts of scientific data (CERN)
- features a C++ interpreter (CLING) i.e. used for rapid and efficient prototyping

Scenarios Use Cases

persistency mechanism for C++ objects



C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart - rootJS





System Model

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

Interface

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



Scenarios Use Cases



Hardware



- Task: encapsulation of ROOT objects and functions
 - ightarrow scanning ROOT structures during initialization
 - → encapsulating objects with heavily nested object structures

Environment Data Interface Scenarios Use Cases System Model

→ introduce (proxy) object cache

Usage

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

Environment Data Interface Scenarios Use Cases

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 Use Cases

Data

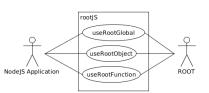
Interface

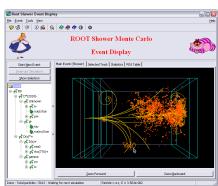
Product interface



Event Viewer









UseROOTGlobal



	Karterune institute or recrinicogy
Use case name	UseROOTGlobal
Participating actor	Initiated by NodeJSApplication; Pro-
instances	cessed by rootJS; Communicates with
	ROOT
Flow of events	
	The NodeJSApplication requests access to a global variable of ROOT. The NodeJSApplication requests to the second of the second
	ProotJS sends a request to the corresponding ROOT variable.
	ROOT returns the requested variable value.
	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.

PSE Purpose Usage Environment Data Interface Scenarios Use Cases System Model

UseROOTObject



Use case name	UseROOTObject
Participating actor	Initiated by NodeJSApplication; Pro-
instances	cessed by rootJS, ProxyObject; Commu-
	nicates with ROOT
Flow of events	
	The NodeJSApplication requests access to a ROOT object by calling a constructor function.
	ProotJS encapsulates the requested ROOT object within a ProxyObject that was created recursively.



PSE Purpose Usage Environment Data Interface Scenarios Use Cases System Model

UseROOTObject



Flow of events

- TootJS stores the created ProxyObject in a cache memory.
- 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.



PSE Purpose

Usage

Environment Data Interface Scenarios Use Cases

UseROOTFunction



Use case name	UseR00TFunction
Participating actor	Initiated by NodeJSApplication; Pro-
instances	cessed by rootJS, ProxyObject; Commu-
	nicates with ROOT
Flow of events	
	The NodeJSApplication requests access to a ROOT function.
	rootJS calls the corresponding ROOT function.
	3 ROOT responds.



UseROOTFunction



- TootJS encapsulates the returned ROOT object within a ProxyObject.
- 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.

Scenarios Use Cases



Environment Data Interface

Usage

UseJIT



Use Case name	UseJIT
Participating actor	Initiated by NodeJSApplication; Pro-
instances	cessed by rootJS, Cling; Communicates
	with ROOT
Flow of events	
	 The NodeJSApplication wants to execute ROOT specific C++ code (given as string) during runtime. rootJS forwards the instructions to Cling.
	Cling evaluates the received instructions using JIT compilation concepts and dynamically modifies the state of ROOT.

PSE Purpose Usage Environment Data Interface Scenarios Use Cases System Model

UseJIT

PSE Purpose

Usage



Flow of events

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

Entry condition	rootJS and Cling have been initialized.
Exit condition	rootJS either confirms the proper ex-
	ecution of the specified instructions
	or forwards thrown exceptions to the
	NodeJSApplication.



Environment Data Interface Scenarios Use Cases



Client application

ROOT framework

TROOT

PSE Purpose Usage Environment Data Interface Scenarios Use Cases System Model





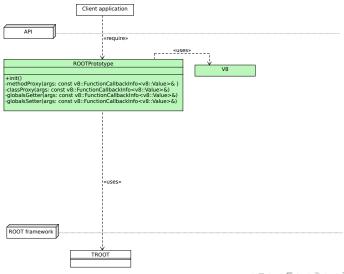


ROOT framework

TROOT







Scenarios Use Cases

System Model

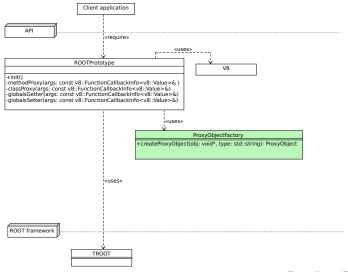


Environment Data Interface

Usage

PSE Purpose

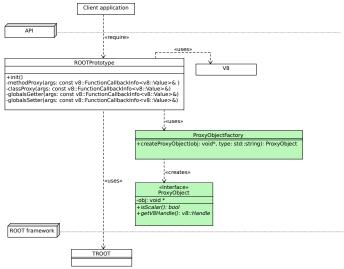




Scenarios Use Cases

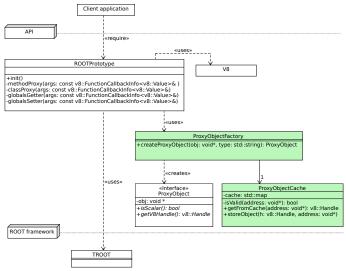
System Model



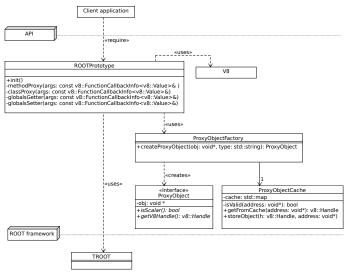


Scenarios Use Cases









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

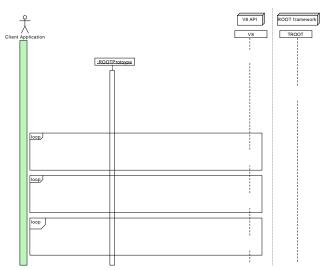
Names

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

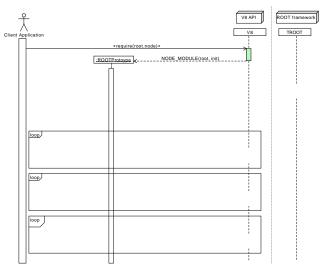


C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart - rootJS

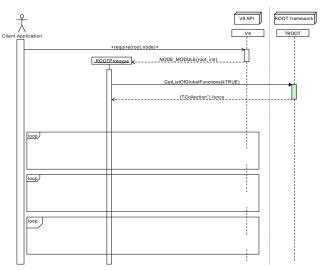




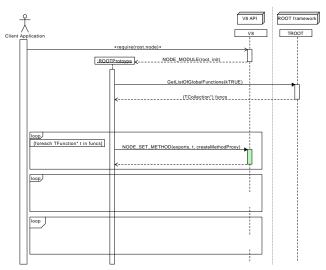






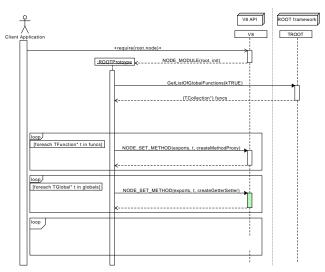








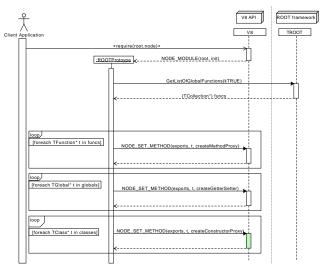






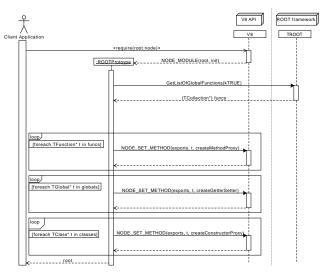
Test Cases Statistics











Call a feature



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

Environment Data Interface Scenarios Use Cases

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

Scenarios Use Cases

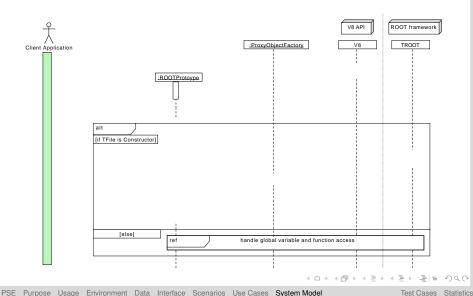
Interface

Call a feature

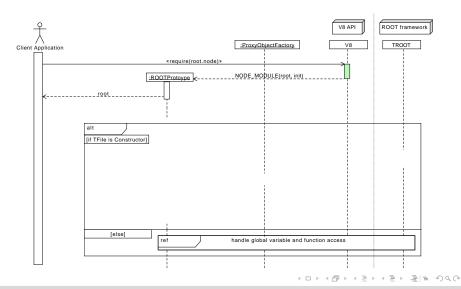


- All features in node are mapped to a proxy method that will be called
- The proxy method will eventually call a root function and pass the result to our ObjectFactory
- By looking at the object type an corresponding v8::Handle will be generated and returned to node
 - If the result is an object this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

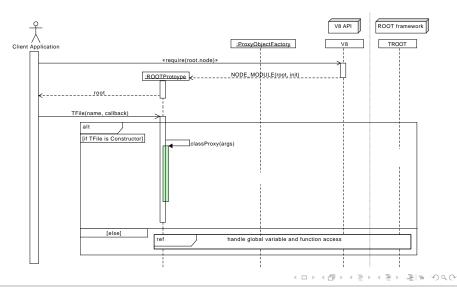




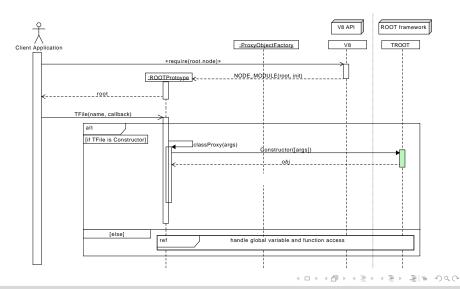




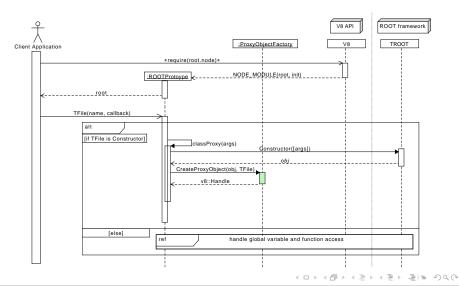




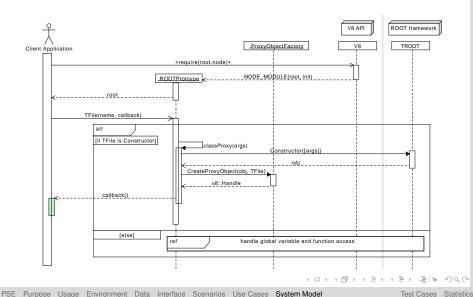




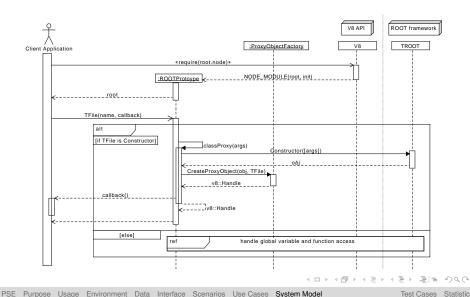












Test Cases

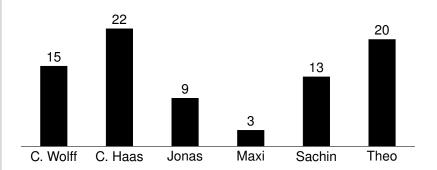


C. Wolff, M. Früh, S. Rajgopal, C. Haas, J. Schwabe, T. Beffart - rootJS

Test Cases Statistics

Merges



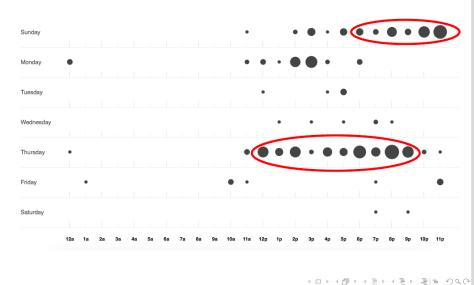




Test Cases Statistics

Punchcard





PSE Purpose Usage Environment Data Interface Scenarios Use Cases System Model Test Cases Statistics

References I

