



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

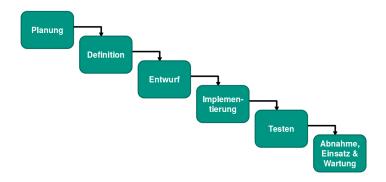


About PSE



Praxis der Softwareentwicklung(PSE) = Software Engineering Practice

- Waterfall model
 - Planning/definition





Purpose



Node.js bindings for ROOT

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

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

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)

Test Cases

4/47

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 \rightarrow web-applications
- Expose processed data and then visualize it locally
- Interact with remote data (i.e. streamed via RPC)
- Accessible on 'unconvential' devices (mobile phones/tablets)

Audience



Scientists (e.g. particle physicists) and Researchers

Interface

- \blacksquare Typical user will know ROOT and JavaScript \to rather technology proficient
- Web-developers



Purpose

Environment

Operating conditions



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

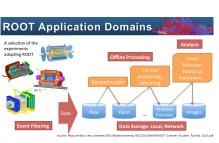
Scenarios

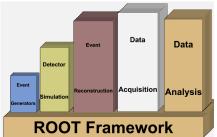
Use Cases

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





December 15, 2015

Use Cases

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

Data

Interface

Scenarios

Purpose

Hardware



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

Data

Interface

Scenarios

⇒ generally negligible hardware requirements of the bindings themselves

Purpose

Test 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

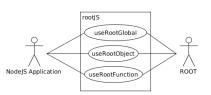
Scenarios

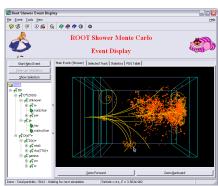
Product interface



Event Viewer









Data

Interface

Scenarios

Use Cases

Purpose

Usage

System Model

UseROOTGlobal



	Karbrune institute or recinicogy
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. rootJS sends a request to the corresponding ROOT variable. ROOT returns the requested variable value.
	The value is passed from rootJS to the NodeJSApplication.



Usage

Data

Interface

Scenarios

UseROOTGlobal



Entry condition	rootJS has been initialized.
Exit condition	The value has been returned to the
	NodeJSApplication.

Interface

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.
	2 rootJS encapsulates the requested ROOT object within a ProxyObject that was created recursively.



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.



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.

Usage

Data

UseROOTFunction



Flow of events

- 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



Purpose

Environment

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

Interface

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.

UseJIT



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.





Client application

ROOT framework

TROOT

Data

Interface



System Model

Test Cases

Scenarios

Use Cases



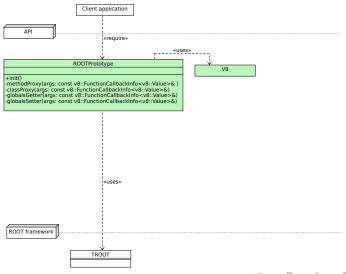


ROOT framework

TROOT





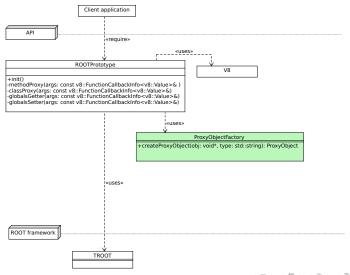


Scenarios

Interface

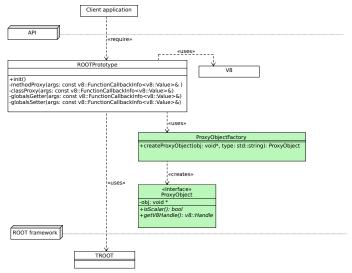
Use Cases



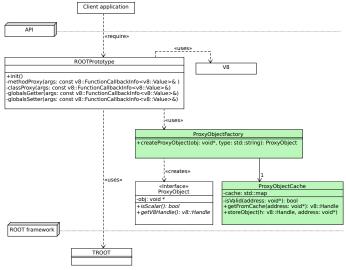


4 ロ > 4 団 > 4 豆 > 4 豆 > 豆 = め 9 Q ()



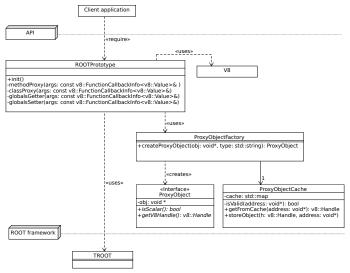






Data





Scenarios



Initialization



- Expose all
 - Global variables
 - Global functions
 - Classes

Scenarios

Use Cases

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

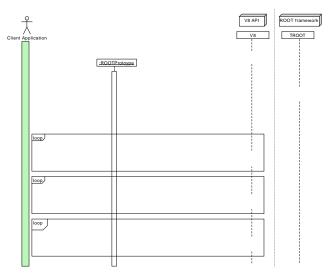
Names

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

Scenarios







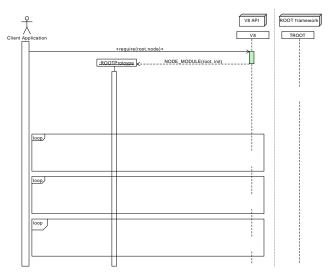
Scenarios

Use Cases

System Model

Test Cases

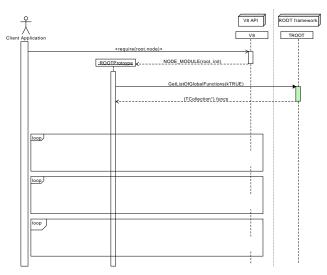




Scenarios

Use Cases

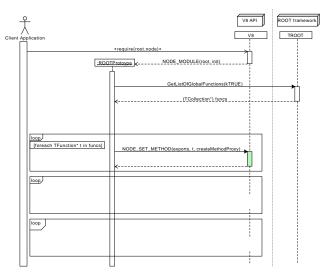




Scenarios

Use Cases



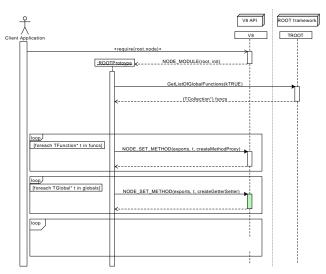


Scenarios

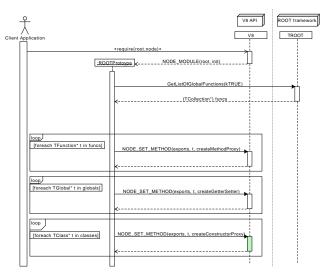
Use Cases

Interface

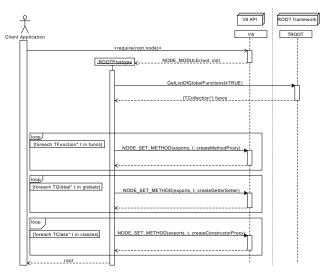












Use Cases

Call a feature



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

Test Cases

Use Cases

Scenarios

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

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

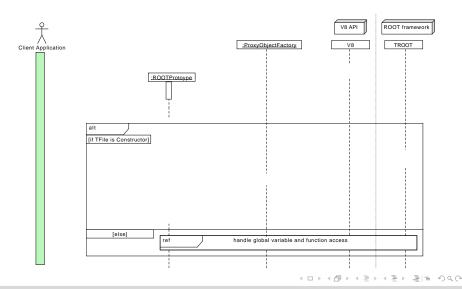
Call a feature



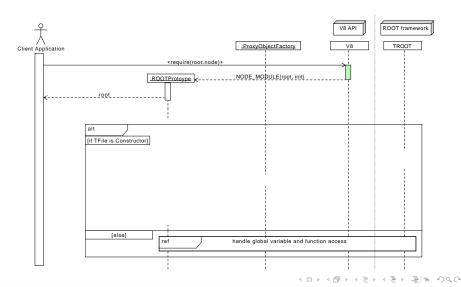
- All features in node are mapped to a proxy method that will be called
- The proxy method will eventually call a root function and pass the result to our ObjectFactory
- By looking at the object type an corresponding v8::Handle will be generated and returned to node
 - If the result is an object this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

December 15, 2015

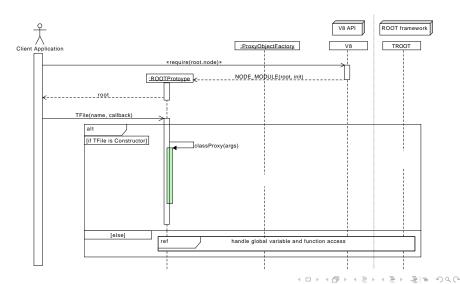




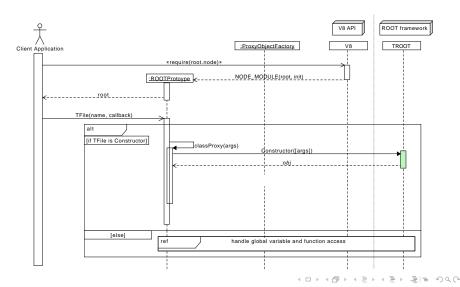












Scenarios

Use Cases

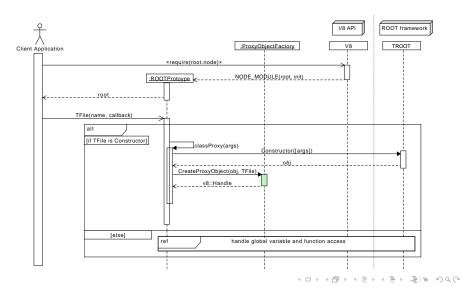
Interface

System Model

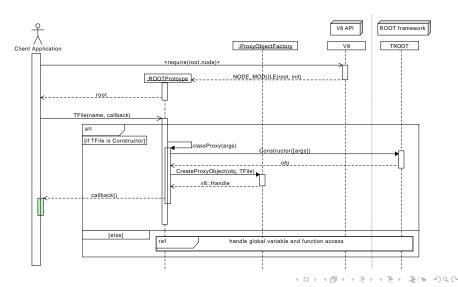
Purpose

Usage









Scenarios

Use Cases

Data

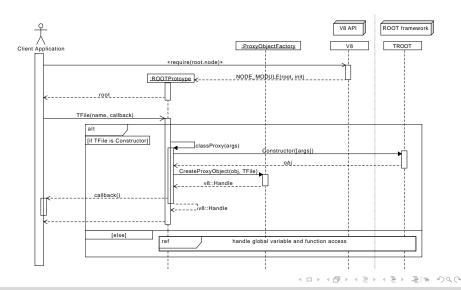
Interface

Purpose

Usage

System Model





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

