Karlsruhe Institute of Technology

rootJS - Design

PSE – Software Engineering Practice

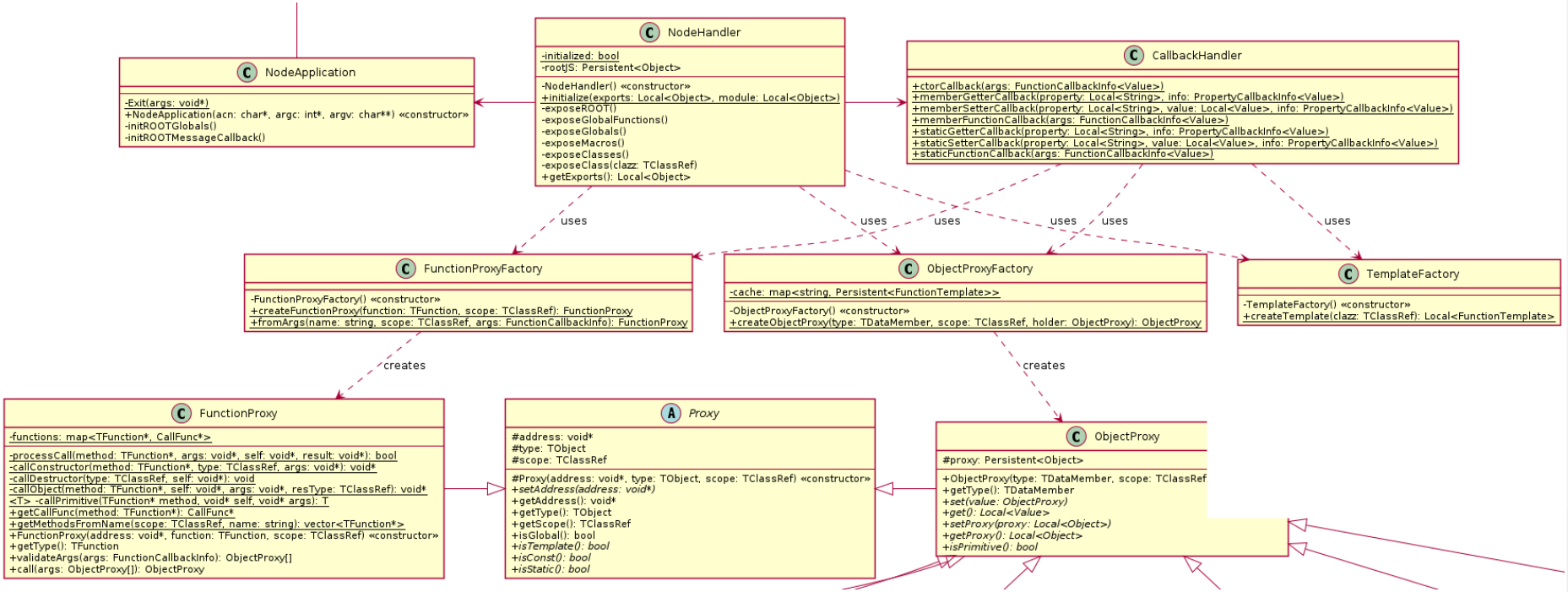
J. Schwabe, T. Beffart, M. Früh, S. Rajgopal, C. Wolff, C. Haas

STEINBUCH CENTRE FOR COMPUTING

Recap

- Create node.js bindings for ROOT
  - Make all ROOT functions avaiable through node.js

Introduction



NodeHandler

- Entry point for node.js App
- Exposes ROOT to node.js
  - Using V8‘s export functionality

```
//in node.js
var root = require(rootJS.node)
//in C++
NODE_MODULE(rootJS, initialize)
```

NodeHandler

-initialized: bool
-rootJS: Persistent<Object>

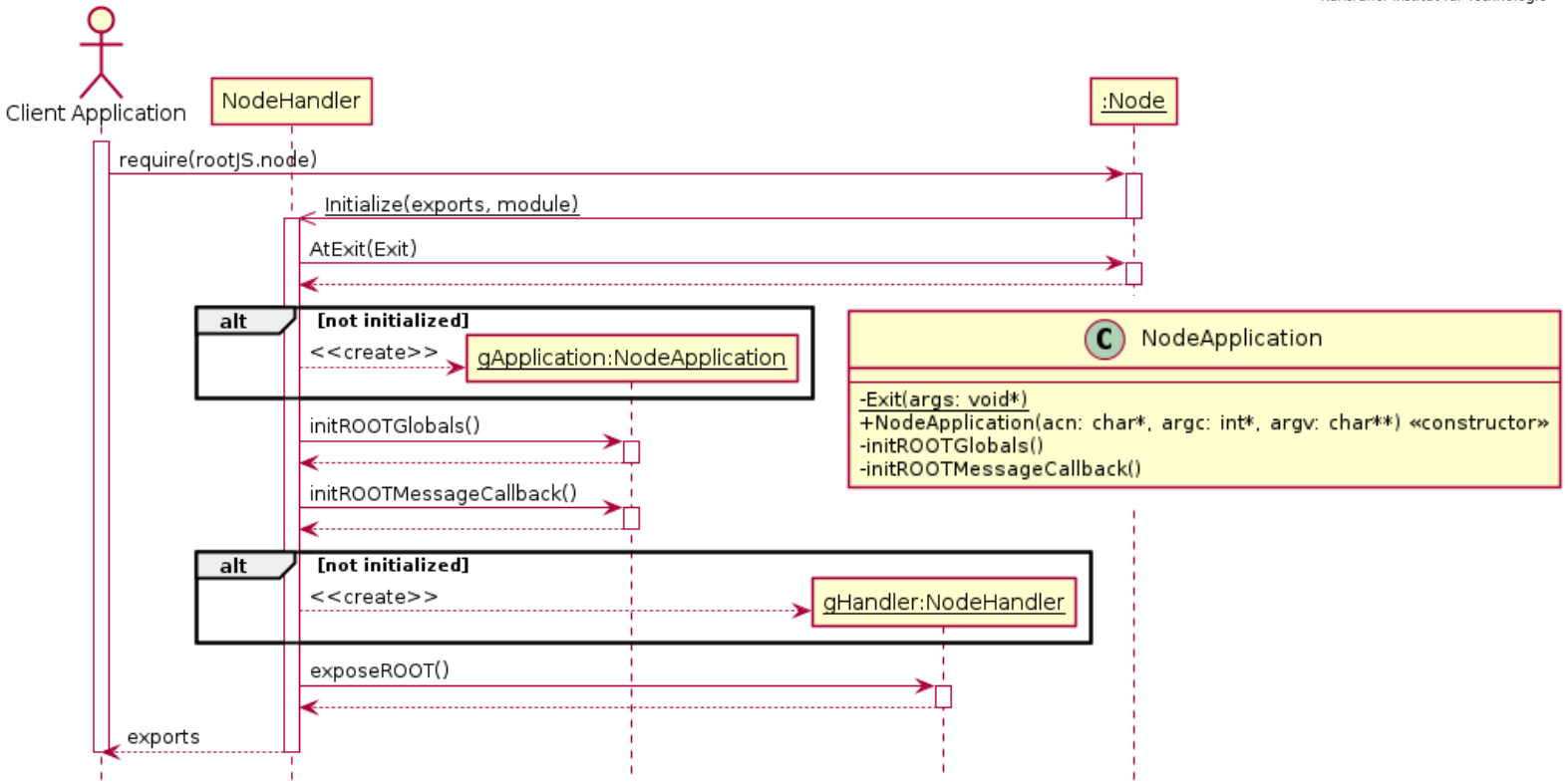
-NodeHandler() «constructor»

+initialize(exports: Local<Object>, module: Local<Object>)

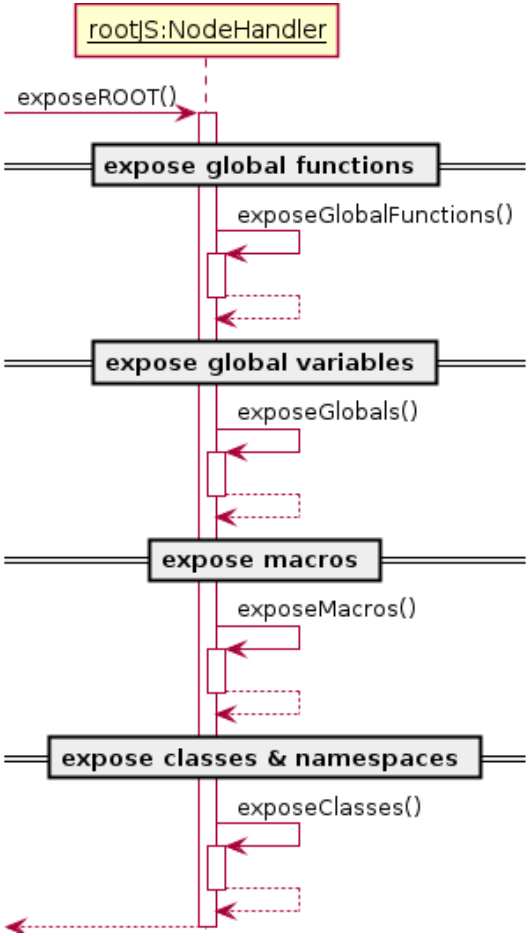
-exposeROOT()
-exposeGlobalFunctions()
-exposeGlobals()
-exposeMacros()
-exposeClasses()
-exposeClass(clazz: TClassRef)

+getExports(): Local<Object>

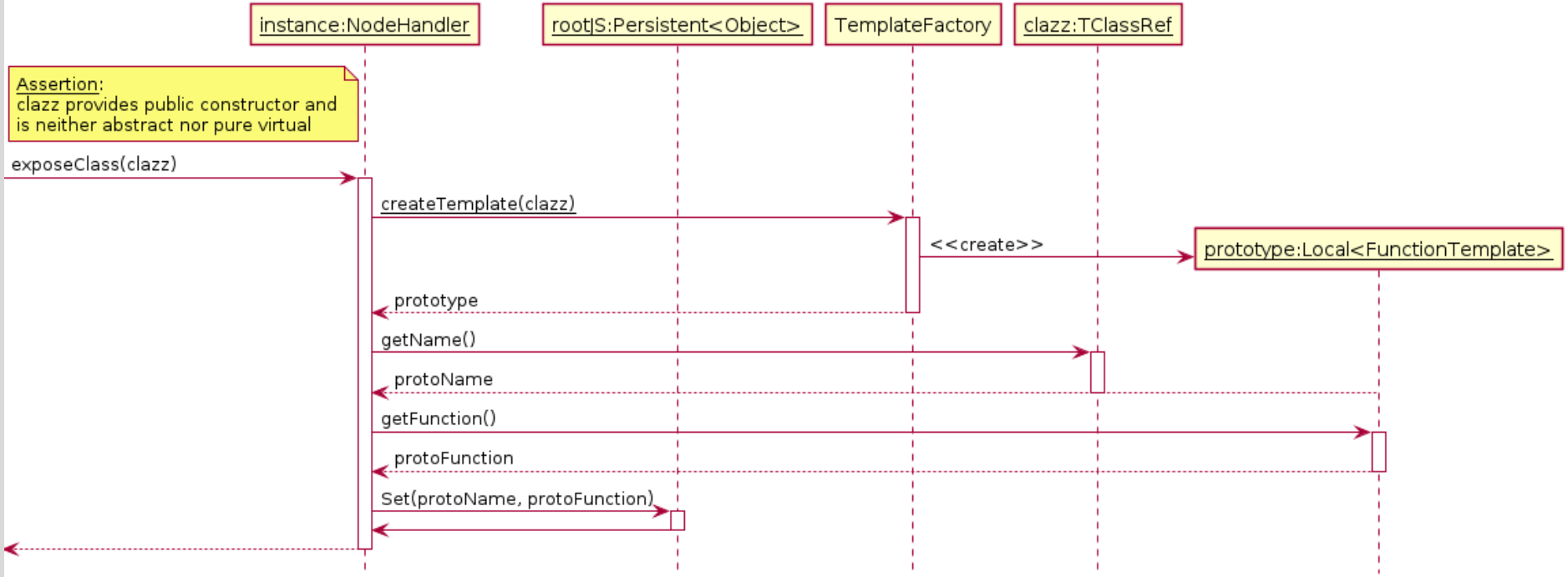
Initialization sequence



Exposure Sequence

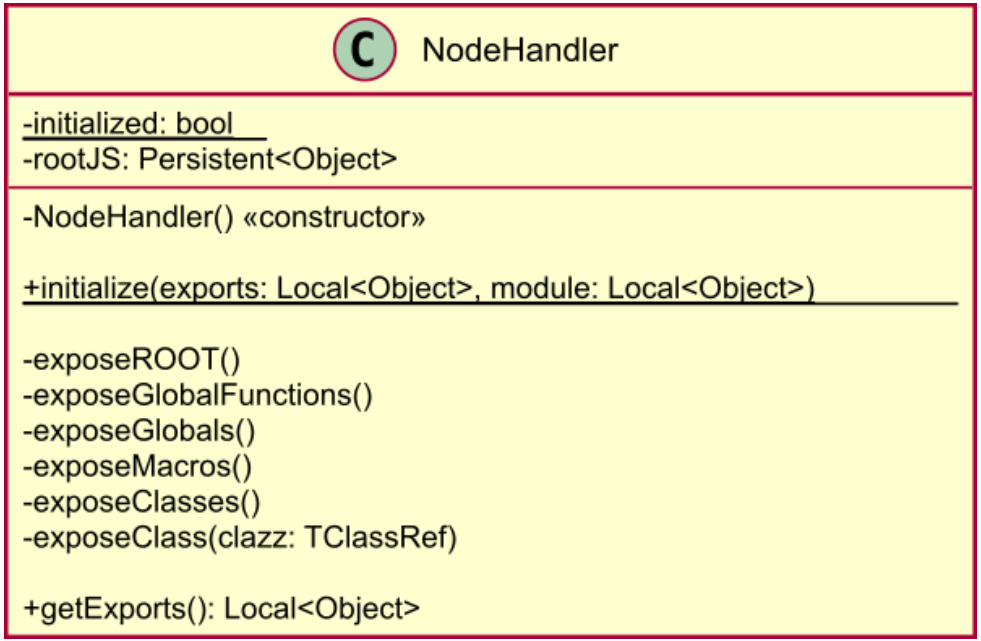


Class Exposure Sequence

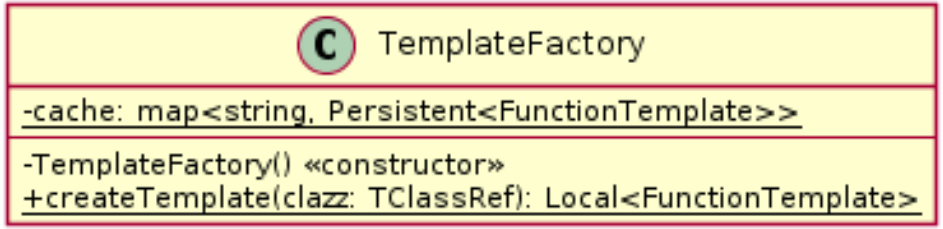


NodeHandler

- Entry point for node.js App
- Exposes ROOT to node.js
  - Using V8‘s export functionality
- After Initialization sequence
  - Static variables available
  - Static methods callable
  - Constructors ready to initialize classes

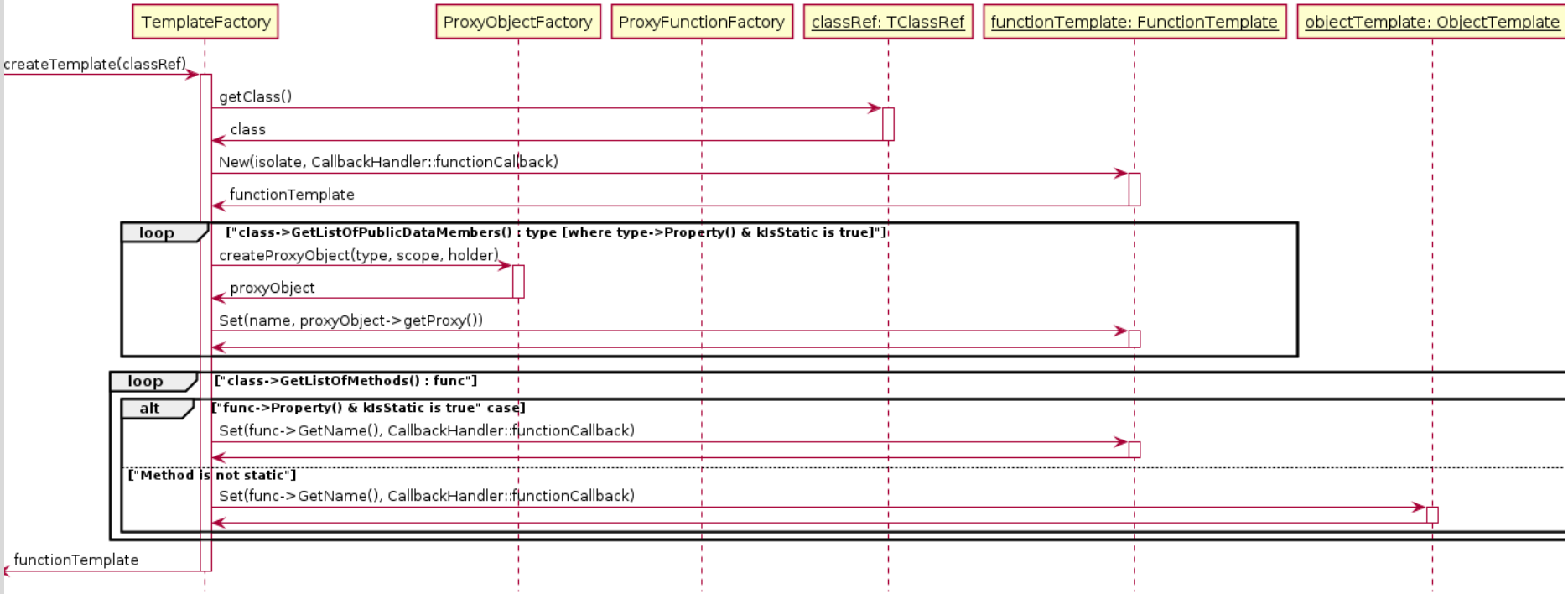


TemplateFactory



- Creates template from reflection of a ROOT class
- Iterates over the class to
  - Create proxies of all static members
  - Create proxies of all methods
- Uses the proxy factories to create those
- Used to export the constructor and the static methods

Template creation sequence



CallbackHandler

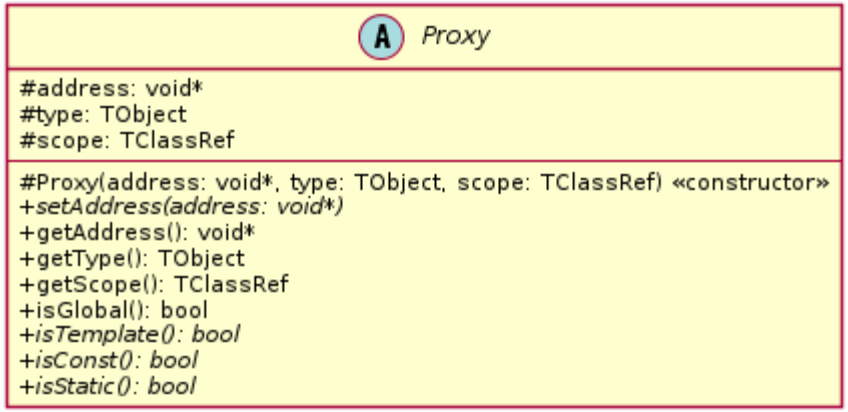
```
CallbackHandler
+ctorCallback(args: FunctionCallbackInfo<Value>)
+memberGetterCallback(property: Local<String>, info: PropertyCallbackInfo<Value>)
+memberSetterCallback(property: Local<String>, value: Local<Value>, info: PropertyCallbackInfo<Value>)
+memberFunctionCallback(args: FunctionCallbackInfo<Value>)
+staticGetterCallback(property: Local<String>, info: PropertyCallbackInfo<Value>)
+staticSetterCallback(property: Local<String>, value: Local<Value>, info: PropertyCallbackInfo<Value>)
+staticFunctionCallback(args: FunctionCallbackInfo<Value>)
```

- The CallbackHandler class gets invoked whenever an encapsulated ROOT function or object is accessed.
- Based on the provided callback information ROOT objects may be manipulated and function calls may be delegated properly .
  - ROOT objects <-> Javascript objects
  - ROOT functions <-> Javascript functions

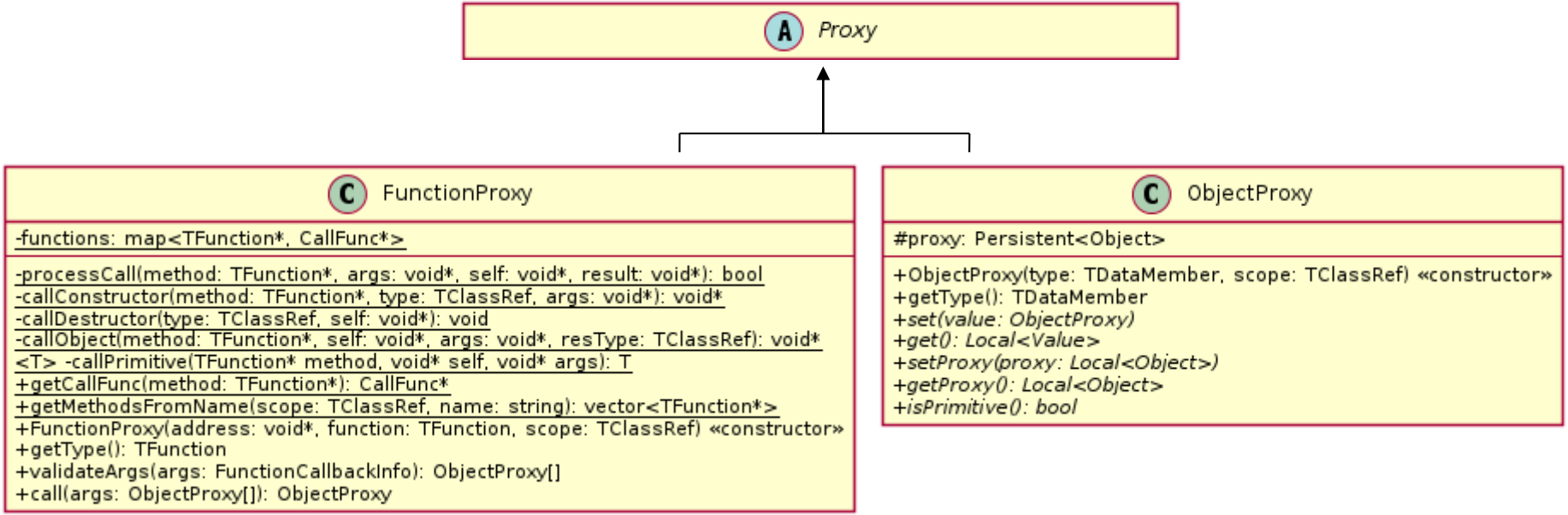
ROOT Proxy

- Unified interface to encapsulate both ROOT objects and functions

- C++ <-> Javascript
- Stored data
  - Memory address
  - Data type
  - Data scope

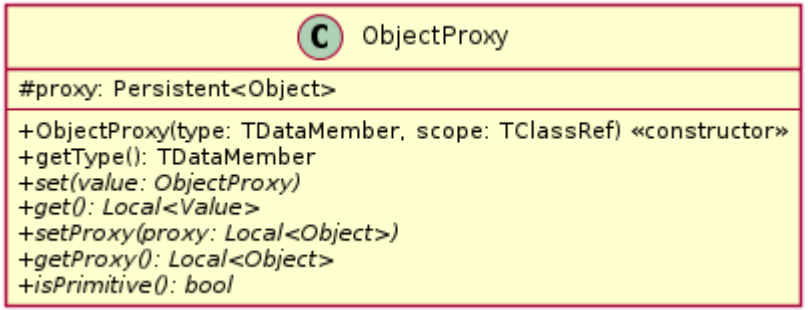


ROOT Proxy


Proxy
FunctionProxy
-functions: map<TFunction*, CallFunc*>
-processCall(method: TFunction*, args: void*, self: void*, result: void*): bool
-callConstructor(method: TFunction*, type: TClassRef, args: void*): void*
-callDestructor(type: TClassRef, self: void*): void
-callObject(method: TFunction*, self: void*, args: void*, resType: TClassRef): void*
<T> -callPrimitive(TFunction* method, void* self, void* args): T
+getCallFunc(method: TFunction*): CallFunc*
+getMethodsFromName(scope: TClassRef, name: string): vector<TFunction*>
+FunctionProxy(address: void*, function: TFunction, scope: TClassRef) «constructor»
+getType(): TFunction
+validateArgs(args: FunctionCallbackInfo): ObjectProxy[]
+call(args: ObjectProxy[]): ObjectProxy
ObjectProxy
#proxy: Persistent<Object>
+ObjectProxy(type: TDataMember, scope: TClassRef) «constructor»
+getType(): TDataMember
+set(value: ObjectProxy)
+get(): Local<Value>
+setProxy(proxy: Local<Object>)
+getProxy(): Local<Object>
+isPrimitive(): bool

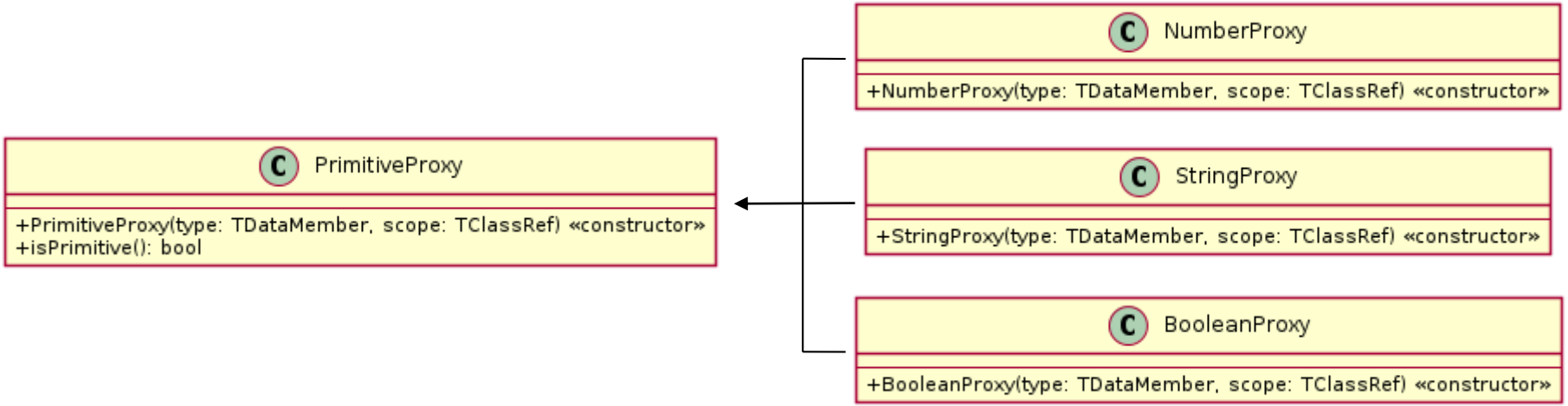
ObjectProxy

- Direct access to C++ object in memory
  - *set(value: ObjectProxy)*
  - *get() : Local<Value>*



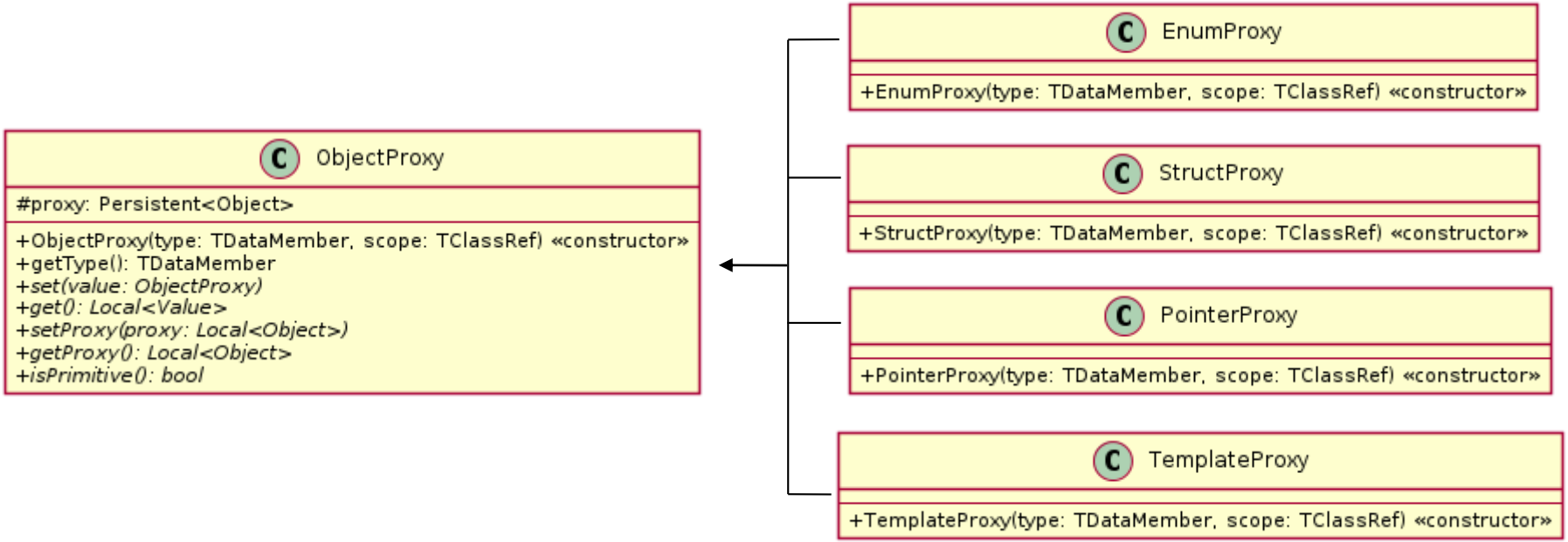
- Hold reference to encapsulating Javascript *proxy* object via internal field
- Differentiate between primitive and non-primitive proxy types

Encapsulating primitives



- Generically cast types using specialized inner classes

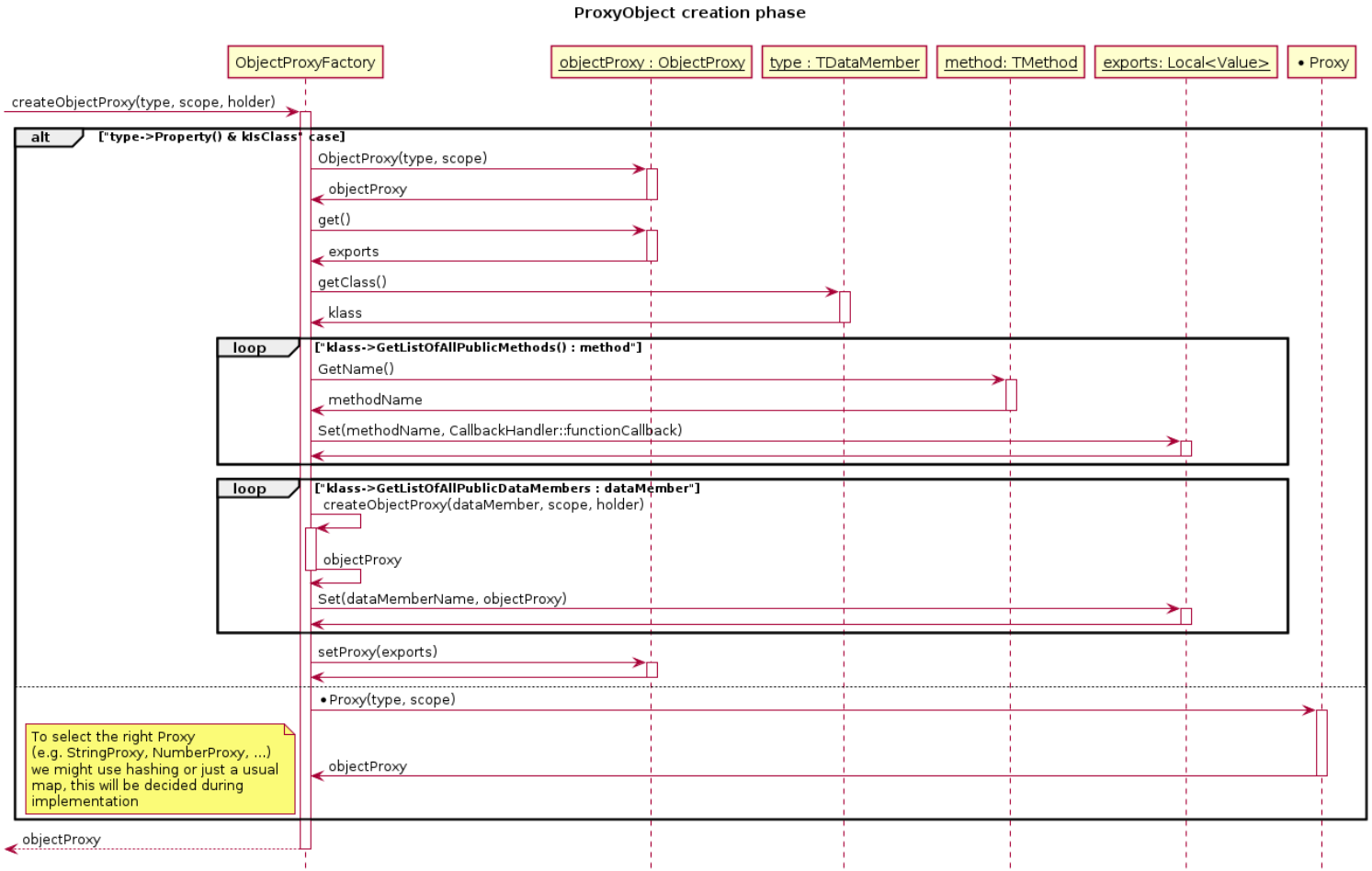
Encapsulating non-primitives



ObjectProxyFactory

```
ObjectProxyFactory
-ObjectProxyFactory() «constructor»
+createObjectProxy(type: TDataMember, scope: TClassRef, holder: ObjectProxy): ObjectProxy
```

- Generate *ProxyObject*‘s for exposed JavaScript objects
  - Recursively process *TDataMembers* until primitive data is reached (depth first search)
  - Use proxied holder for offset calculation of base address
  - Caching mechanisms are used to speed up the generation during run time
    - Also resolves cyclic dependencies

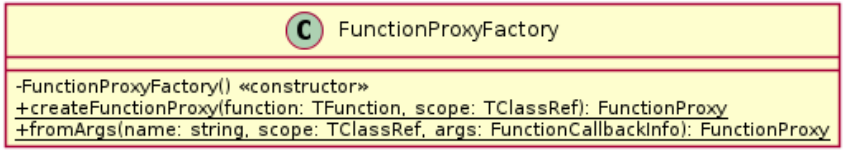

ProxyObject creation phase
ObjectProxyFactory
objectProxy : ObjectProxy
type : TDataMember
method: TMethod
exports: Local<Value>
• Proxy
createObjectProxy(type, scope, holder)
alt
["type->Property() & kIsClass" case]
ObjectProxy(type, scope)
objectProxy
get()
exports
getClass()
klass
loop
["klass->GetListOfAllPublicMethods() : method"]
GetName()
methodName
Set(methodName, CallbackHandler::functionCallback)
loop
["klass->GetListOfAllPublicDataMembers : dataMember"]
createObjectProxy(dataMember, scope, holder)
objectProxy
Set(dataMemberName, objectProxy)
setProxy(exports)
• Proxy(type, scope)
To select the right Proxy
(e.g. StringProxy, NumberProxy, ...)
we might use hashing or just a usual
map, this will be decided during
implementation
objectProxy
objectProxy

FunctionProxy

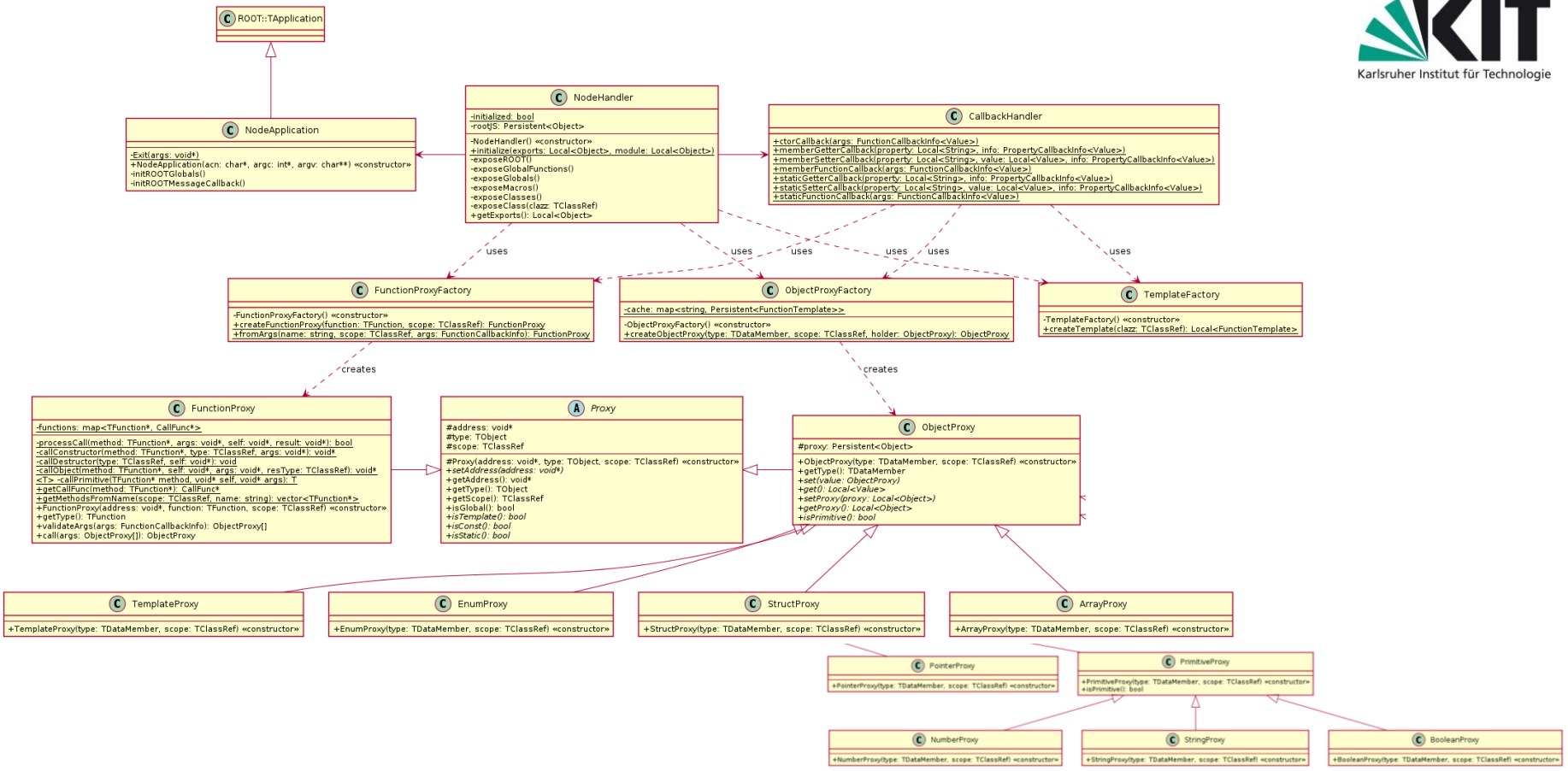
```
FunctionProxy
-functions: map<TFunction*, CallFunc*>
-processCall(method: TFunction*, args: void*, self: void*, result: void*): bool
-callConstructor(method: TFunction*, type: TClassRef, args: void*): void*
-callDestructor(type: TClassRef, self: void*): void
-callObject(method: TFunction*, self: void*, args: void*, resType: TClassRef): void*
<T> -callPrimitive(TFunction* method, void* self, void* args): T
+getCallFunc(method: TFunction*): CallFunc*
+getMethodsFromName(scope: TClassRef, name: string): vector<TFunction*>
+FunctionProxy(address: void*, function: TFunction, scope: TClassRef) «constructor»
+getType(): TFunction
+validateArgs(args: FunctionCallbackInfo): ObjectProxy[]
+call(args: ObjectProxy[]): ObjectProxy
```

- Makes ROOT callables accessible
- CallFunc* points to the wrapped function in storage
  - Is provided by Cling
- *validateArgs* checks arguments passed by V8 in the FunctionCallbackInfo and converts them to ObjectProxies

FunctionProxyFactory



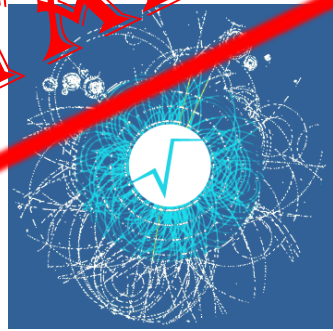
- Creates function proxies
- *fromArgs* to find the correct method in case of overloading


ROOT::TApplication
NodeApplication
-Exit(args: void*)
+NodeApplication(acn: char*, argc: int*, argv: char**) «constructor»
-initROOTGlobals()
-initROOTMessageCallback()
NodeHandler
-initialized: bool
-rootJS: Persistent<Object>
-NodeHandler() «constructor»
+initialize(exports: Local<Object>, module: Local<Object>)
-exposeROOT()
-exposeGlobalFunctions()
-exposeGlobals()
-exposeMacros()
-exposeClasses()
-exposeClass(clazz: TClassRef)
+getExports(): Local<Object>
CallbackHandler
+ctorCallback(args: FunctionCallbackInfo<Value>)
+memberGetterCallback(property: Local<String>, info: PropertyCallbackInfo<Value>)
+memberSetterCallback(property: Local<String>, value: Local<Value>, info: PropertyCallbackInfo<Value>)
+memberFunctionCallback(args: FunctionCallbackInfo<Value>)
+staticGetterCallback(property: Local<String>, info: PropertyCallbackInfo<Value>)
+staticSetterCallback(property: Local<String>, value: Local<Value>, info: PropertyCallbackInfo<Value>)
+staticFunctionCallback(args: FunctionCallbackInfo<Value>)
uses
uses
uses
uses
uses
uses
FunctionProxyFactory
-FunctionProxyFactory() «constructor»
+createFunctionProxy(function: TFunction, scope: TClassRef): FunctionProxy
+fromArgs(name: string, scope: TClassRef, args: FunctionCallbackInfo): FunctionProxy
ObjectProxyFactory
-cache: map<string, Persistent<FunctionTemplate>>
-ObjectProxyFactory() «constructor»
+createObjectProxy(type: TDataMember, scope: TClassRef, holder: ObjectProxy): ObjectProxy
TemplateFactory
-TemplateFactory() «constructor»
+createTemplate(clazz: TClassRef): Local<FunctionTemplate>
creates
creates
FunctionProxy
-functions: map<TFunction*, CallFunc*>
-processCall(method: TFunction*, args: void*, self: void*, result: void*): bool
-callConstructor(method: TFunction*, type: TClassRef, args: void*): void*
-callDestructor(type: TClassRef, self: void*): void
-callObject(method: TFunction*, self: void*, args: void*, resType: TClassRef): void*
<T> -callPrimitive(TFunction* method, void* self, void* args): T
+getCallFunc(method: TFunction*): CallFunc*
+getMethodsFromName(scope: TClassRef, name: string): vector<TFunction*>
+FunctionProxy(address: void*, function: TFunction, scope: TClassRef) «constructor»
+getType(): TFunction
+validateArgs(args: FunctionCallbackInfo): ObjectProxy[]
+call(args: ObjectProxy[]): ObjectProxy
Proxy
#address: void*
#type: TObject
#scope: TClassRef
#Proxy(address: void*, type: TObject, scope: TClassRef) «constructor»
+setAddress(address: void*)
+getAddress(): void*
+getType(): TObject
+getScope(): TClassRef
+isGlobal(): bool
+isTemplate(): bool
+isConst(): bool
+isStatic(): bool
ObjectProxy
#proxy: Persistent<Object>
+ObjectProxy(type: TDataMember, scope: TClassRef) «constructor»
+getType(): TDataMember
+set(value: ObjectProxy)
+get(): Local<Value>
+setProxy(proxy: Local<Object>)
+getProxy(): Local<Object>
+isPrimitive(): bool
TemplateProxy
+TemplateProxy(type: TDataMember, scope: TClassRef) «constructor»
EnumProxy
+EnumProxy(type: TDataMember, scope: TClassRef) «constructor»
StructProxy
+StructProxy(type: TDataMember, scope: TClassRef) «constructor»
ArrayProxy
+ArrayProxy(type: TDataMember, scope: TClassRef) «constructor»
PointerProxy
+PointerProxy(type: TDataMember, scope: TClassRef) «constructor»
PrimitiveProxy
+PrimitiveProxy(type: TDataMember, scope: TClassRef) «constructor»
+isPrimitive(): bool
NumberProxy
+NumberProxy(type: TDataMember, scope: TClassRef) «constructor»
StringProxy
+StringProxy(type: TDataMember, scope: TClassRef) «constructor»
BooleanProxy
+BooleanProxy(type: TDataMember, scope: TClassRef) «constructor»

Non-Blocking

libuv

IMPLEMENTATION DETAIL



ROOT

Data Analysis Framework