Fraunhofer FOKUS

Kaiserin-Augusta-Allee 31

10589 Berlin

Tel: +49 30 3463-7000

info@fokus.fraunhofer.de

OpenRide Mobile Client – Javascript Modularization Concepts

Javascript is not an Object Oriented Programming language, but complex javascript-controlled web applications – like the OpenRide Mobile Web Client – still need some kind of modularization to allow following object oriented design patterns to some extend in order to build generic, changeable and extendable code.

Using the JavaScript prototype chain would be a common (build-in) way to allow polymorphic structures. A slim client design is more efficient (which is especially important in the mobile domain) and the complexity of functionality and state in the OpenRide Mobile Web Client currently does not need inheritance. Therefore prototype chaining is only implemented to establish the namespace for packaging:

in initcontroller.js the namespace 'fokus.openride.mobclient.controller.modules' is established as a protoype chain. This allowes hanging in new modules anywhere in the namespace (e.g. the initcontroller itself as well as the serverconnector are registered under fokus.openride.mobclient.controller). This does not only allow packaging but also shields the modules, which can only be accessed with the correct namespace.

The OpenRide modules make use of the concepts of encapsulation via Javascript Closures. They allow setting the visibility of variables and functions to private or public.

The following code snippet does nothing meaningful but shows the basic structure of a closure, as we use it for module creation. It also shows how a new module is registered in the namespace:

```
//moulename is prefixed with the namespace it should be registered in
fokus.openride.mobclient.example.testmodule = function(){
        //private variable, can only be used inside the module
```

```
	var x = 0;


	//private method, can only be used inside the module
	var privateFunc = function(y){
		alert("x + y  is "+ (x+y));
	};


	return {
		//public variable, can be accessed from outside
		publicField : 'use and set me from outside',

		//public method, can be accessed from outside
		magicFunc : function(y){
		  privateFunc(y);
		  x++;
		},


		magicFunc2 : function(y){
		  this.magicFunc(y);
		  alert("x is"+x);
		}
	}
}();
```

The 2 braces in the last line let the whole module code get executed, when the javascript file gets loaded into the browser. From this point on an object for this module exists in memory.

Everything in the "return {};" statement is publicly available from outside the module. All private variables and functions before the return statement are kept in memory for usage in the public part, but they cannot directly be accessed from outside the module.

note: *the public part of the module is a JSON-like structure, a comma-separated list of variables/functions.*

note: *magicFunc2 shows, that you can only access a public function within the same module using "this."*

note: *using "this." can lead to problems with the Javascript context. For example, if magicfunc2 would be used within an AJAX-callback, "this." would refer to the object holding the callback method, but not to the module anymore. To avoid hard to determine runtime-errors which could arise from this, please keep the "this"-reference in another variable for use in the public functions. In this case using a private variable "var testmoduleTHIS = this;" in combination with the according call "testmoduleTHIS .magicFunc(y);" in magicFunc2 would solve the issue.*

When a javascript module is loaded by the browser, its code gets executed and the module object is kept in memory. If module A contains a reference to another module B, B must already exist, when A's script file is read by the browser, so B's script must be loaded before A's script. Cross-references (B references A and A references B) have to be avoided because of this. While most Browsers load script in the order in which the <script …/> tags appears in HTML, the W3C HTML standard does not guarantee any order for script-loading, so this behavior should not be relied on. To address this issue, the OpenRide Mobile Web Client uses code compression to merge the whole Javascript into 1 file before sending it to the client browser.