The Revealing Module Pattern

Now that we're a little more familiar with the module pattern, let’s take a look at a slightly improved version - Christian Heilmann’s Revealing Module pattern.

The Revealing Module pattern came about as Heilmann was frustrated with the fact that he had to repeat the name of the main object when we wanted to call one public method from another or access public variables.  He also disliked the Module pattern’s requirement for having to switch to object literal notation for the things he wished to make public.

The result of his efforts was an updated pattern where we would simply define all of our functions and variables in the private scope and return an anonymous object with pointers to the private functionality we wished to reveal as public.

An example of how to use the Revealing Module pattern can be found below:

[?](https://addyosmani.com/resources/essentialjsdesignpatterns/book/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | var myRevealingModule = (function () {            var privateVar = "Ben Cherry",              publicVar = "Hey there!";            function privateFunction() {              console.log( "Name:" + privateVar );          }            function publicSetName( strName ) {              privateVar = strName;          }            function publicGetName() {              privateFunction();          }              // Reveal public pointers to          // private functions and properties            return {              setName: publicSetName,              greeting: publicVar,              getName: publicGetName          };        })();    myRevealingModule.setName( "Paul Kinlan" ); |

The pattern can also be used to reveal private functions and properties with a more specific naming scheme if we would prefer:

[?](https://addyosmani.com/resources/essentialjsdesignpatterns/book/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | var myRevealingModule = (function () {            var privateCounter = 0;            function privateFunction() {              privateCounter++;          }            function publicFunction() {              publicIncrement();          }            function publicIncrement() {              privateFunction();          }            function publicGetCount(){            return privateCounter;          }            // Reveal public pointers to          // private functions and properties           return {              start: publicFunction,              increment: publicIncrement,              count: publicGetCount          };        })();    myRevealingModule.start(); |

**Advantages**

This pattern allows the syntax of our scripts to be more consistent. It also makes it more clear at the end of the module which of our functions and variables may be accessed publicly which eases readability.

**Disadvantages**

A disadvantage of this pattern is that if a private function refers to a public function, that public function can't be overridden if a patch is necessary. This is because the private function will continue to refer to the private implementation and the pattern doesn't apply to public members, only to functions.

Public object members which refer to private variables are also subject to the no-patch rule notes above.

As a result of this, modules created with the Revealing Module pattern may be more fragile than those created with the original Module pattern, so care should be taken during usage.