Stijn de Witt's Blog

Coder by day, Programmer by night

Enums in Javascript

Posted on January 26, 2014 by Stijn de Witt

TL;DR: Enums in Javascript

Define your enum like so:

```
var SizeEnum = {
  SMALL: 1,
  MEDIUM: 2,
  LARGE: 3,
};
```

Then use it like so:

```
var mySize = SizeEnum.SMALL;
```

If you want the enum values to hold properties, you can add them to an extra object:

```
var SizeEnum = {
  SMALL: 1,
  MEDIUM: 2,
  LARGE: 3,
  properties: {
    1: {name: "small", value: 1, code: "S"},
    2: {name: "medium", value: 2, code: "M"},
    3: {name: "large", value: 3, code: "L"}
  }
};
```

Then use it like so:

```
var mySize = SizeEnum.MEDIUM;
var myCode = SizeEnum.properties[mySize].code; // myCode == "M"
```

Background information

The format described above is the result of quite some time of thinking about enums in Javascript. It tries to combine the best of both worlds of using primitives as the enum values (safe for de/serializing) and using objects as the values (allows properties on the values). Read further to learn how I came to this format.

Rediscovering enums

I recently stumbled onto a question (http://stackoverflow.com/questions/287903/enums-in-javascript) on StackOverflow that I had answered myself a couple of years back and did some more thinking about it after reading some of the comments and decided that this topic was worth an article.

So what was the question I hear you ask?

What is the best way to write enums in Javascript?

First of all, before answering this question, we have to have a look at what an enum is and what it means to write one in Javascript. So let us look at a definition of enum:

What is an enum?

> *In computer programming, an enumerated type (also called enumeration or enum [..]) is a data type consisting of a set of named values called elements, members or enumerators of the type. The enumerator names are usually identifiers that behave as constants in the language. A variable that has been declared as having an enumerated type can be assigned any of the enumerators as a value.*
> *–Wikipedia: Enumerated type (http://en.wikipedia.org/wiki/Enumerated_type)*

And a good example often beats a formal definition:

```
enum WeekDay = {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY,
SUNDAY};
```

So, summarizing: An enum is a type restricting variables to one value from a predefined set of constants. In the example above, `WeekDay` is an *enum* and `MONDAY`, `TUESDAY` etc are the constants in the set, also called the *enumerators*. If we declare a variable as

```
WeekDay payDay;
```

..we would be able to assign it any of the constants `MONDAY`, `TUESDAY` etc, up to and including `SUNDAY`, but not something else like `12` or `"labour day"`.

…which brings us to a problem.

It can’t be done in Javascript

Javascript is a weakly typed (http://en.wikipedia.org/wiki/Weak_typing) language, which means you don’t declare variables to have a specific type beforehand. In Java (which is strongly typed) you might write:

```
int i; // declares a variable named i which may hold integer values.
```

If you would then later on attempt to assign it a string:

```
i = "Hello World";
```

…the compiler would give you an error and refuse to produce a program.
Not so in Javascript:

```
var i;
i = 10;
i = "Hello World";
i = 3.1415;
i = true;
i = ['my', 'array'];
i = {look: 'at', my: 'object'};
```

As you can see we declare a variable `i` (with the `var` keyword) but it’s runtime type is not restricted in any way. We can assign it any value we like. The wording on Wikipedia is a bit awkward, but if you read between the lines of this sentence *“A variable that has been declared as having an enumerated type can be assigned any of the enumerators as a value.”*, it suggests that any of the enumerators can be assigned, **but nothing else!**. So it restricts the values that can be assigned to the variable. In Javascript we just can never do this. So we can’t write real enums. We can however *simulate* them to get some of the convenience they offer in languages like C. But keep in mind, it’s just a simulation that gives us some syntax suger.

Writing ‘enums’ in Javascript

I come from Java, which is a strong typed language, but incidentally also did not have enums for a long time. So Java programmers came up with a few different kind of solutions to simulate them.

Constants, often i.c.w. a naming convention

Just list the constants at the top of the class where you will use them:

```
public static final int DAYS_MONDAY = 0;
public static final int DAYS_TUESDAY = 1;
// ..
public static final int DAYS_SUNDAY = 6;
```

Class with simple constants

Like the example above, but put the constants in a dedicated class to promote re-use:

```
public class DaysEnum {
  public static final int MONDAY = 0;
  public static final int TUESDAY = 1;
  // ..
  public static final int SUNDAY = 6;
}
```

Class with instances as constants

The most advanced simulation also seems to have inspired the final implementation of real enums in the Java language itself. It uses a dedicated class with a private constructor (meaning no instances can be created apart from within the class itself) and uses instances of itself as the enumerators:

```
public class DaysEnum {
  private DaysEnum() {}
  public static final DaysEnum MONDAY = new DaysEnum();
  public static final DaysEnum TUESDAY = new DaysEnum();
  // ..
  public static final DaysEnum SUNDAY = new DaysEnum();
}
```

The last version which uses instances of the enum as the enumerators is very elegant. It makes the simulated enums truly typesafe. With the other two variations, variables that would hold an enum value would just be ints:

```
int payDay = DAYS_FRIDAY; // variation 1
int payDay = DaysEnum.FRIDAY; // variation 2
```

It would still be possible to assign a completely wrong value, such as 128 to such an enum. In contrast, the third variation actually restricts the values that can be assigned to the enumerators listed in the enum:

```
DaysEnum payDay = DaysEnum.FRIDAY; // ok
DaysEnum payDay = 128; // compiler error
```

As an added bonus, the third enum pattern allows us to add extra fields, for example to hold the name of the day, and even methods (`isWeekendDay()` for example) to the enumerators. So, coming from Java and knowing these patterns, my answer (http://stackoverflow.com/questions/287903/enums-in-javascript/2383215#2383215) on StackOverflow suggested the Javascript version of this third variation. It's still my highest scoring answer on StackOverflow, but I don't really stand by it anymore. So let me explain to you why and then, finally, show you how I personally think you should write enums in Javascript to get the maximum benefits and as little gotcha's as possible.

So what's the Javascript version of alternative 3 and what is wrong with it?

This is what the third variation would look like when coded in Javascript:

```
var DaysEnum = {
  MONDAY: {}, // optionally you can give the object properties and methods
  TUESDAY: {},
  // ..
  SUNDAY: {}
};
```

But as said, I now no longer recommend this style. **Don't use it**.

Why not?

Because, as jcollum (http://stackoverflow.com/users/30946/jcollum) pointed out to me in the comments of some of the other answers on the SO thread, this technique gives issues when data is serialized (sent over the wire). To understand why this is, let's look at what happens when we assign one of the enumerators from DaysEnum to a field of an object:

```
var myObject = {
  payDay: DaysEnum.FRIDAY
};

var yesterday = DaysEnum.THURSDAY, today = DaysEnum.FRIDAY;
if (yesterday == myObject.payDay)
  alert("Yesterday was pay day... but not today...");
else if (today == myObject.payDay)
  alert("Today is pay day! Yippie!!!");
else
  alert("Neither yesterday nor today are pay days... I'm broke!");
```

Ok, so far so good. This alerts `"Today is pay day! Yippie!!!"` as we might expect. But let's see what happens when we serialize myObject to JSON and then deserialize it back:

```
var serialized = JSON.stringify(myObject);
alert("serialized myObject: " + serialized);
var deserializedObject = JSON.parse(serialized);
if (yesterday == deserializedObject.payDay)
```

```
  alert("Yesterday was pay day... but not today...");
else if (today == deserializedObject.payDay)
  alert("Today is pay day! Yippie!!!");
else
  alert("Neither yesterday nor today are pay days... I'm broke!");
```

This ends up alerting `"Neither yesterday nor today are pay days... I'm broke!"`. The reason this happens is that upon deserialization `JSON.parse` creates a **new** object as the value for `payDay`. This new object is not the same as `DaysEnum.FRIDAY`, so the comparisons all fail and we end up in the last `else` branch.

There are probably ways to work around this, but is it worth it? I would say not. And being able to serialize and deserialize the enums is too important to just ignore this issue. So that's why I advice against using this pattern. Instead, go with the one derived from variation 2, which does not have this issue:

```
var DaysEnum = {
  MONDAY: "monday",
  TUESDAY: "tuesday",
  // ..
  SUNDAY: "sunday"
};
```

(or you could use numbers instead of strings for the values, both will work equally fine)

Let's check that this can be serialized safely:

```
var myObject = {
  payDay: DaysEnum.FRIDAY
};

var serialized = JSON.stringify(myObject);
alert("serialized myObject: " + serialized);
var deserializedObject = JSON.parse(serialized);
if (yesterday == deserializedObject.payDay)
  alert("Yesterday was pay day... but not today...");
else if (today == deserializedObject.payDay)
  alert("Today is pay day! Yippie!!!");
else
  alert("Neither yesterday not today are pay days... I'm broke!");
```

This works as expected, alerting `"Today is pay day! Yippie!!!"`.

But I want my fields and methods!

Yes that's a shame isn't it? The ability to give the enumerators fields and methods was really attractive. We could do this:

```
var SizeEnum = {
  SMALL: {name: "small", value: 1, code: "S"},
  MEDIUM: {name: "medium", value: 2, code: "M"},
  LARGE: {name: "large", value: 3, code: "L"},
};
```

But, as said, this does not survive serialization/deserialization… So what now?

Well, we can just add the properties to an extra object:

```
var SizeEnum = {
  SMALL: 1,
  MEDIUM: 2,
  LARGE: 3,
  properties: {
    1: {name: "small", value: 1, code: "S"},
    2: {name: "medium", value: 2, code: "M"},
    3: {name: "large", value: 3, code: "L"}
  }
};
```

Then we could access the enum properties like this:

```
var mySize = SizeEnum.MEDIUM;
var myCode = SizeEnum.properties[mySize].code; // myCode == "M"
```

…admittedly less elegant but hey that's life. We got to make hard choices. I'd say that the ability for an enum value to survive serialization/deserialization is much more important than having smart properties on the instance.

Object.freeze

In typesafe languages that have enums, these types are understood to be constant. The set of values does not change, nor do the constant values themselves. In Javascript however, we may overwrite any of the constants at any time, or add new ones to the set etc. If you want to prevent this, you may want to have a look at Object.freeze (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze):

> *The Object.freeze() method freezes an object: that is, prevents new properties from being added to it; prevents existing properties from being removed; and prevents existing properties, or their enumerability, configurability, or writability, from being changed. In essence the object is made effectively immutable. The method returns the object being frozen.*

Sounds exactly like what we need right?

Because not all browsers support it (http://kangax.github.io/es5-compat-table/) yet, you should test for it's existence before using it:

```
if (Object.freeze)
  Object.freeze(DaysEnum);
```

(Thanks to Artur Czajka (http://stackoverflow.com/users/572370/artur-czajka) for pointing out Object.freeze)

So, there you have it. Enums in Javascript. Now go forth and code!

This entry was posted in Javascript, Programming, Programming patterns and tagged enums, Javascript, patterns. Bookmark the permalink.

26 responses to "*Enums in Javascript*"

Jason says:
January 29, 2014 at 19:48
Hi Stijn,

I like your blog. It's well-formed, concise, and logical.

I have had to work with this issue at work this week. For me, the point of the enum is that it nicely prevents this:

var DaysEnum = {
MONDAY: "monday",
TUESDAY: "tuesday",
// ..
SUNDAY: "sunday"
};

var day = DaysEnum.MONDAY.
…
…
if (day === "monday") // do some stuff

This still allows us to compare enums with other things successfully. You could always provide serialization and deserialization methods when this is important:

var DaysEnum = {
MONDAY: {},
TUESDAY: {},
// ..
SUNDAY: {},

```
serialize : function( day ) {
if ( day === MONDAY ) return “monday”;
else if ( day === TUESDAY ) return “tuesday”;
…
},

deserialize : function() {
if ( day === “monday” ) return MONDAY;
else if ( day === “tuesday” ) return TUESDAY;
…
}
};
```

Now,

```
var day = DaysEnum.MONDAY;
var dayToSave = DaysEnum.serialize( day );
```

and later

```
var someDay = DaysEnum.deserialize( dayToSave );
```

and finally this means that

```
someDay === DaysEnum.MONDAY
```

is true! Additionally, we run into no bugs or shortcuts where

```
someDay === “monday”
```

creeps in.

It was useful for me. I hope it can be a useful alternative for others.

Cheers!
Jason

stijndewitt says:
January 31, 2014 at 02:13
Hi Jason, thanks for reading my blog.

Yes, providing (de)serialization methods could do the trick. But you don’t always have control over the code that does the (de)serialization, so you might not be able to make that code use those methods. And even when you do have control, those enums still become special fields that you will have to carefully remember to treat specially when it comes to serializing or deserializing an object.

With serialization becoming so common now with the increased use of ajax and json in ‘web 2.0’ websites I personally feel that it’s not worth it, but that depends on the apps you are building I guess.

Lucas Pereira Caixeta says:
July 24, 2014 at 18:53
Hey, can I translate this article to portoguese and put on my blog? I´m from Brazil.

stijndewitt says:
July 25, 2014 at 00:52
Hi Lucas, that's a great idea! You are free to translate my article as long as you link back to the original. Let me know once you're done and I will post a link to your article as well.

Santiago says:
September 5, 2014 at 15:38
useful, thanks!

g2-754970d3251f37967021c7bc09fed8ff says:
September 5, 2014 at 19:53
I think the code below mimics what you're going for in terms of enums, however, I like it as it has that cleaner feel that enum declarations other languages give us *and* it doesn't suffer from having to match enum values in more than one place. People can change the internals for how properties are interpreted and stored to suit their own particular coding circumstance or style.

```
var SizeEnum = {
SMALL: 1,
MEDIUM: 2,
LARGE: 3,
properties: {
1: {name: "small", value: 1, code: "S"},
2: {name: "medium", value: 2, code: "M"},
3: {name: "large", value: 3, code: "L"}
}
};

var SizeEnum2 = mkenum({ SMALL: 1, MEDIUM: 2, LARGE: 3});
var SizeEnum3 = mkenum({ SMALL: [1, "small", "S"], MEDIUM: [2, "medium",
"M"], LARGE: [3, "large", "L"] });

/* mkenum
*
* Loop through hash collecting inverse mappings and optional other
* bits into property hash which gets added at end. If optional
* properties present, replace with enum value. Both optional
* property converters (see if test, "v instanceof Array"), can be
* whatever you like.
*
*/
function mkenum(seed)
{
var p = {};

console.log("Seed := " + seed);

for (k in seed) {
var v = seed[k];
console.log("V is Array? " + (v instanceof Array));

if (v instanceof Array)
p[(seed[k]=v[0])] = { value: v[0], name: v[1], code: v[2] };
else
p[v] = { value: v, name: k.toLowerCase(), code: k.substring(0,1) };
}
seed.properties = p;

return Object.freeze ? Object.freeze(seed) : seed;
}

console.log(JSON.stringify(SizeEnum));
console.log(JSON.stringify(SizeEnum2));
console.log(JSON.stringify(SizeEnum3));
```

stijndewitt says:
September 5, 2014 at 21:48
Wow looks like a nice piece of work! I think I will be using this in the future 🙂

Ranando King says:
September 10, 2014 at 17:52
For a while now I've looked for a good way to handle enums in Javascript the way I'm accustom to in other languages. Sadly even what you've got here doesn't fit the bill. It fails at the most critical point: constraining the value of a variable to the values available in the enum type.

Consider enums in C/C++:

enum Color { Red, Green, Blue }; //Defines a new enum type named Color
Color color; //Defines a new enum variable that only takes Color values.

It's that second line that's missing in every enum attempt I've seen to date. Since with the code presented in this article, all you're doing is copying objects, it makes sense that (de)serialization would break the comparison operator. The deserialized object is a new object and not the same as the one that was serialized, even though the value is the same.

https://drive.google.com/file/d/0B3E82DN8Gph_alNyNy13QnptbTg/edit?usp=sharing

That link points to an Enum object I've created that tackles this issue in a different way. You'd use it like this:

var DaysEnum = new Enum("Monday", ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", Sunday"]); //Defines a new enum type
var day1 = new DaysEnum("Wednesday"); //Defines a new enum variable
var day2 = new DaysEnum(DaysEnum.Wednesday);
var day3 = new DaysEnum(2);

day1.name == day2.name; // => true
day1.value == day2.value; // => true
day1 == day2; // => false. Don't do this. They're different objects.

day3.value == day1.value; // => true. If the enum elements are passed as an array, their values are ordinals from 0.
day3.name == new DaysEnum(JSON.parse(JSON.stringify(day3))).name; // => true

day1.name = "Sunday";
day1.value == 6; // => true
day2.value = DaysEnum.Friday;
day2.name == "Friday"; // => true
day3.name = "foo"; // => throws!
day3,value = "bar"; // => throws!
day3.value = 4; // => ok!
day3.name; // => Friday

You can also use object notation for the second Enum parameter. Do that ONLY if you want to specify your own name:value pairs for the enum. The value part of the pair can be anything, including objects. This class preserves as much of the enum semantics as I could remember and javascript would allow.

Ranando King says:
September 10, 2014 at 18:16
Just ran a few more tests against object initialization:

var TestEnum = new Enum("a", {a: {foo: "foo", bar: "bar"}, b: {foo: "snafu", bar: "fubar"}});
var test = new TestEnum();
var test2 = new TestEnum();
test == test2
// => false
test.name == test2.name
// => true
test.value == test2.value
// => true
test.value = TestEnum.b
test2.name = "b"
test.value == test2.value
// => true
test2.value = {foo: "snafu", bar: "fubar"}
// => "Element value '[object Object]' is not a member of this Enum!"
test2.value = new TestEnum({foo: "snafu", bar: "fubar"}).value;
// => "Parameter '[object Object]' is not a name or value match for any member of this Enum!"
test2.value = new TestEnum({name: "b", value: {foo: "snafu", bar: "fubar"}}).value;
test.value == test2.value
// => true

Ashok Kumar says:
October 19, 2014 at 11:18
Hi,
If the topic is tooooooo lengthy, then people cannot spare time to concentrate on your blog. Please target the point while writing blog.
I got nice help from the following URL, but NOT from your blog.
http://stravid.com/en/cleaner-javascript-code-with-enums/

I know that you won’t publish my comment, but I targeted you while writing this comment.

codeedog says:
October 19, 2014 at 16:14
Hey, Ashok,

People’s experiences vary. I learned more about javascript reading his blog post and in the process was inspired to write my own little enum creator library, teaching me even more.

To be fair to Stijn, he has found that the simpler enum approaches, like the one you pointed out, don’t meet his needs. He stated this quite clearly in the blog post you didn’t both to read.

12. Pingback: How to: Enums in JavaScript? | SevenNet
13. Pingback: Solution: Enums in JavaScript? #dev #it #computers | Technical information for you
14. Pingback: Fixed Enums in JavaScript? #dev #it #asnwer | Good Answer

mohanradhakrishnan says:
January 19, 2015 at 14:19
Is it possible to create your final solution dynamically ? I have to display a code, some text and a check box. There are several such rows. I also want to add a selected property. Can I create such an enum and update the ‘selected’ property ? Otherwise I have create objects for each row ?

Stijn de Witt says:
January 22, 2015 at 09:55
Hi mohanradhakrish, thanks for commenting.

Yes it is possible… Javascript is a dynamic language so it can be done. However I would advice against it. Enums are supposed to represent a type. So in the Weekday example the enum represents types of days: Sunday, Monday, Tuesday etc. However what you need aren’t types but instances. For your purpose I would just create objects for each rows, like so:

```
var rows = []; // Create an empty array
for all rows {
var obj = {
code: ‘Code for this row’,
text: ‘Text for this row’,
selected: false
};
rows.push(obj);
}
// Later in the code, user selected row 3…
rows[2].selected = true;
```

If you want to go really fancy you can create a ‘class’ by using prototype inheritance. It depends on your needs whether that is worth it.

Jon W says:
March 29, 2015 at 20:51

I found this blog because I am getting so used to enums in other languages (I like you hava a java background), and I will use your advice as my model. Of course I still have work to do to fully understand it 🙂 Thanks much!!

Brian Cummings says:
May 28, 2015 at 21:45
Hi Everyone – Great article and great comments. Thanks a lot!!

RE: The reply from g2-754970d3251f37967021c7bc09fed8ff on September 5, 2014 at 19:53. One minor point – to return an entirely 'frozen' object you need to 'freeze' each element in the 'properties' collection…

Stijn de Witt says:
May 29, 2015 at 16:58
Thanks for your comment Jon! Glad it helped you.
Good point about freeze Brian. If one day I find the time I might roll all this into a little library function that does it all for you.

Amila says:
January 12, 2016 at 05:18
Good post,
small question, Is there any way to mapping attribute set without iterating.
my use case: i got some json array.i want to map that attribute values in to another separate attribute set. can enum use for that. i don't want to iterate the json array object.

Stijn de Witt says:
January 12, 2016 at 13:23
Hi Amila,

I'm not sure I understand your question… Do you mean to get the names / values of all the properties of an object?

In case you have an object like this:

```
var WeekDays = {
MONDAY: 0,
TUESDAY: 1,
// …
SUNDAY: 6
};
```

You can get the names 'MONDAY', 'TUESDAY' etc into an arry using Object.keys():

```
var dayNames = Object.keys(WeekDays);
```

To get all the values does require a loop… Or you could write your own utility function:

```
function objectValues(obj) {
var keys = Object.keys(obj);
var result = [];
for (var i = 0; i < keys.length; i++) {
```

```
result.push(obj[keys[i]]);
}
return result;
}
```

then use it like this:

```
var dayConstants = objectValues(WeekDays);
```

You talk about having a JSON array… I'm not sure what that means exactly, but assuming you have an array of objects, like so:

```
var people = [
{
name: 'Alice',
age: 37
}, {
name: 'Bob',
age: 23,
}, {
name: 'Charlie',
age: 26
}
];
```

You can use Array.map to select say their names:

```
var names = people.map(function(person){
return person.name;
});
```

Good luck and let me know if you got it to work!

Diego Marin says:
November 3, 2016 at 16:25
Why not using something like below and use SizeEnum.equals(o1, o2) when it's necessary to compare Enum objects?

```
var SizeEnum = {
SMALL: {name: “small”, value: 1, code: “S”},,
MEDIUM: {name: “medium”, value: 2, code: “M”},
LARGE: {name: “large”, value: 3, code: “L”},

this.prototype.equals = function (o1, o2) {
//code using value property to compare, for example
};
};
```

Stijn de Witt says:
November 5, 2016 at 11:41

@Diego Marin
Thans for your comments, I love getting those!

Yes, that would be one solution. But with that solution, the problem is you will have to make sure that other developers are aware of this. It's a difference with the way things 'normally' work, so it's going to be something you'll have to document carefully. And even then; people don't read the docs so there are going to be issues reported about it probably.

But, if you are the only one using this code, or if you are on a small team and everyone understands this, then yes it would definitely work.

Naresh Bhatia says:
January 1, 2017 at 18:10
Hi Stijn,

This is a great post! Thanks for taking the time to write out your thoughts.

I have a quick question: while your recommended approach solves the serialization issue, it seems to create another – I can no longer enumerate all the keys easily. Object.keys(SizeEnum) will not only include real keys, but also 'properties' as a key. Am I missing something?

Here's my use case: I want to specify menu items as an enumeration. In my render function I would like to iterate through my enumeration and render each menu item.

Forrest Akin says:
January 31, 2017 at 00:27
Hey Naresh, I believe the best solution to your problem is using Object.defineProperty to set 'properties' as a non-enumerable property – good luck!

Stijn de Witt says:
February 1, 2017 at 11:56
Thanks for your comments guys. You raise a good issue Naresh. When I wrote the above code, I did not take nice behavior with Object.keys into account. Forrest's solution should work for that.

Blog at WordPress.com.